Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the mansucript number!)

A comparison of Monte Carlo tree search and rolling horizon optimization for large scale dynamic resource allocation problems *

Dimitris Bertsimas
Operations Research Center and Sloan School of Management, Massachusetts Institute of Technology; 77 Massachusetts Avenue, Cambridge MA 02139; dbertsim@mit.edu, http://www.mit.edu/~dbertsim/

J. Daniel Griffith
Lincoln Laboratory, Massachusetts Institute of Technology; 244 Wood Street, Lexington MA 02420; dan.griffith@ll.mit.edu

Vishal Gupta
Department of Data Sciences and Operations, Marshall School of Business, University of Southern California; 3670 Trousdale Parkway, Los Angeles CA 90089; guptavis@usc.edu, http://www-bcf.usc.edu/~guptavis/

Mykel J. Kochenderfer
Department of Aeronautics and Astronautics, Stanford University; 496 Lomita Mall, Stanford CA 94305; mykel@stanford.edu, http://mykel.kochenderfer.com

Velibor V. Mišić
Operations Research Center, Massachusetts Institute of Technology; 77 Massachusetts Avenue, Cambridge MA 02139; vvmisic@mit.edu, http://www.mit.edu/~vvmisic/

Robert Moss
Lincoln Laboratory, Massachusetts Institute of Technology; 244 Wood Street, Lexington MA 02420; robert.moss@ll.mit.edu

Dynamic resource allocation (DRA) problems constitute an important class of dynamic stochastic optimization problems that arise in a variety of important real-world applications. DRA problems are notoriously difficult to solve to optimality since they frequently combine stochastic elements with intractably large state and action spaces. Although the artificial intelligence and operations research communities have independently proposed two successful frameworks for solving dynamic stochastic optimization problems—Monte Carlo tree search (MCTS) and rolling horizon optimization (RHO), respectively—the relative merits of these two approaches are not well understood. In this paper, we adapt both MCTS and RHO to two problems – a problem inspired by tactical wildfire management and a classical problem involving the control of queueing networks – and undertake an extensive computational study comparing the two methods on large scale instances of both problems in terms of both the state and the action spaces. We show that both methods are able to greatly improve on a baseline, problem-specific heuristic. On smaller instances, the MCTS and RHO approaches perform comparably, but the RHO approach outperforms MCTS as the size of the problem increases for a fixed computational budget.

Distribution A: Public Release

* This work is sponsored by the Assistant Secretary of Defense for Research and Engineering, ASD(R&E), under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

1. Introduction

Dynamic resource allocation (DRA) problems are problems where one must assign resources to tasks or requests over some finite time horizon. Many important real-world problems can be cast as DRA problems, including applications in air traffic control (Barnhart et al. 2003, Bertsimas and Stock Patterson 1998), network revenue management (see, e.g., Talluri and van Ryzin 2004, for an overview), scheduling (Bertsimas et al. 2014, Erdelyi and Topaloglu 2010) and logistics, transportation and fulfillment (Acimovic and Graves 2012, Powell et al. 2002). DRA problems are notoriously difficult to solve for two reasons. First, DRA problems typically exhibit stochasticity, i.e., the requests to be processed may arrive randomly according to some stochastic process that, itself, depends on where resources are allocated. Second, DRA problems exhibit extremely large state and action spaces, making solution by traditional dynamic programming methods infeasible (Bellman 1957, Powell 2011). A number of scientific communities, particularly within artificial intelligence and operations research, have sought more sophisticated techniques for addressing DRA and other dynamic stochastic optimization problems.

Within the AI community, one approach for dynamic stochastic optimization problems that has received increasing attention in the last 15 years is a method known as Monte Carlo tree search (MCTS) (Coulom 2007, Browne et al. 2012). In any dynamic stochastic optimization problem, one can represent the possible trajectories of the system—the state at each decision epoch and the actions taken at those epochs—as a tree, where the root represents the initial state. In MCTS, one iteratively builds an approximation to this tree and uses it to inform the choice of action. In our opinion, notwithstanding some recent viewpoints (Domshlak and Feldman 2013), MCTS's effectiveness stems from two key features: 1) bandit upper confidence bounds (see Auer et al. 2002, Kocsis and Szepesvári 2006) can be used to balance exploration and exploitation in learning, 2) application-specific heuristics and knowledge can be used to customize the base algorithm (Browne et al. 2012). Moreover, the MCTS algorithm is very flexible and can easily be tailored to a variety of problems. Indeed, the only technical requirement for implementing MCTS is a generative model that, given a state and an action at a given decision epoch, generates a new state for the next epoch and an immediate reward received. This flexibility makes MCTS particularly attractive as a general purpose methodology.

Most importantly, MCTS has been extremely successful in a number of applications, particularly in designing expert computer players for difficult games such as Go (Gelly and Silver 2011, Lee et al. 2009, Enzenberger et al. 2010), Hex (Arneson et al. 2009, 2010), Kriegspiel (Ciancarini and Favini 2010), and Poker (Rubin and Watson 2011). Although we think it is fair to say that MCTS is the one of the top performing, general purpose algorithms for this class of games, we observe that games like Go and Hex are qualitatively very different from DRAs. Namely, unlike

typical DRA problems, the state of these games does not evolve stochastically, and, furthermore, the size of the feasible action space is often much smaller than for typical DRA problems. For example, in the instances of Go studied in Gelly and Silver (2011), the action branching factor is at most 81, whereas in the DRA instance we consider, a typical branching factor is approximately 230 million (cf. Eq. (13)). While there have been some studies that apply MCTS to probabilistic problems (Eyerich et al. 2010) and problems with large action spaces (Couëtoux et al. 2011), there is relatively little experience with MCTS in problems of this nature, which motivates the study of MCTS in the context of DRA.

On the other hand, within the operations research community, the study of DRAs has proceeded along different lines. A prominent stream of research is based upon mathematical optimization (MO). In contrast to MCTS which only requires access to a generative model of the stochastic system, MO approaches model the dynamics of the system *explicitly* via a constrained optimization problem. The solution to this optimization problem then yields a control policy for the system. This paradigm has a long history within the dynamic control literature (see, e.g., Bertsekas 1995, for an overview).

In the special case of dynamic resource allocation problems, there are a number of competing proposals to incorporate the stochastic evolution including robust optimization / control (Bertsimas et al. 2011, Ben-Tal et al. 2009, Nilim and El Ghaoui 2005, Grieder et al. 2003) and chance constrained optimization (Charnes and Cooper 1959). In what follows, however, we focus on a third proposal, sometimes called *rolling horizon optimization* (RHO). Specifically, we replace uncertain parameters in a MO formulation with their expected values and periodically re-solve the optimization problem for an updated policy as the true system evolves. Variants of this paradigm are known by many names in the literature including fluid approximation (Gallego and van Ryzin 1994, Avram et al. 1995), certainty equivalent control (Bertsekas 1995, Chapt. 6) or model predictive control (Bemporad 2006). These (re-)optimization frameworks are well-known to have excellent practical performance in application domains like queueing (Chen and Yao 1993, Shah and Wischik 2012) and network revenue management (Ciocan and Farias 2012), and in some special cases, additionally enjoy strong theoretical guarantees on performance (e.g., Ciocan and Farias 2012, Gallego and van Ryzin 1994, Jasin and Kumar 2012, Maglaras and Meissner 2006, Reiman and Wang 2008, Talluri and van Ryzin 1998).

In any case, regardless of the particular proposal for incorporating stochasticity, the widespread use and success of RHO approaches for DRAs contrasts strongly with a lack of computational experience with MCTS for DRAs. Furthermore, the two methods differ philosophically. MCTS as a solution approach involves directly simulating the *true* system and efficiently searching through the tree of state-action trajectories. In contrast, RHO as a solution approach involves first constructing

an *approximation* of the true system and then solving an optimization problem based on this approximation to determine a policy for controlling the *true* system; this policy is generally not guaranteed to be optimal for the true system.

In this paper, we aim to understand the relative merits of both the above MCTS and RHO approaches by applying them to two challenging DRA problems:

1. **Tactical wildfire management.** In this problem, the decision maker must control the spread of a fire on a discrete grid (representing a wildland area) by deploying a limited number of suppression resources to specific locations on this grid. Our interest in this problem as a benchmark problem stems primarily from the computational difficulty of the underlying DRA. In particular, each cell on the grid may be burning or not burning, resulting in an exponentially large number of states of the system, while the allocation decision involves choosing a subset of the burning cells to apply suppression resources to, resulting in an exponentially large number of actions that may be taken at each decision epoch. Aside from its computational difficulty, this problem is also of practical importance. The severity of wildland fires has been steadily increasing in recent years, particularly in the southwestern US (Gorte 2013), and as a result, US federal government spending on wildfire management has also been increasing, amounting to $3.5 billion in 2013 (Bracmort 2013). Suppression costs are only part of the total cost of wildfire; combined costs of loss of property, damage to the environment and loss of human life are estimated to be many times larger (Western Forestry Leadership Coalition 2010). Consequently, there has been renewed interest in models for realtime decision support tools to assist in tactical wildfire management, namely how best to utilize fire suppression resources to contain and extinguish a fire.

2. **Queueing network control.** In this problem, the decision maker is in control of a network of servers that serve jobs of different classes. Jobs of certain classes may arrive exogenously, and . The problem of queueing network control is to decide which job class each server should process at each time so as to minimize the average long run number of jobs in the system. In this problem, the state of the system is encoded by the number of jobs of each class and as such is exponential in the number of job classes. The action at each stage is to decide which class each server should service, and as such the size of the action space also grows in an exponential fashion with the number of servers. Given the combinatorial nature of the underlying state and action spaces, this problem constitutes a challenging DRA. More significantly, this problem is known to be *EXP-complete* (Papadimitriou and Tsitsiklis 1999), which is a strong certificate of its computational intractability. At the same time, queueing networks and the problem of optimally controlling them arise in a number of domains such as manufacturing systems (Buzacott and Shanthikumar 1993), communication networks (Bertsekas and Gallager 1992), computer systems (Harchol-Balter 2013) and neuroscience (Mišić et al. 2014).

We summarize our contributions as follows:

1. We develop an MCTS-based approach for the tactical wildfire management problem and the queueing network control problem. To the best of our knowledge, this represents the first application of MCTS to challenging DRA problems motivated by real-world applications. Towards this end, we combine a number of classical features of MCTS, such as bandit upper confidence bounds, with new features such as double progressive widening (Couëtoux et al. 2011). For the tactical wildfire management problem, we also propose a novel action generation approach to cope with the size of the state and action spaces of the DRA.

2. We propose a mathematical optimization formulation for the tactical wildfire management problem that approximates the original discrete and stochastic elements of the MDP by suitable continuous and deterministic counterparts. Although this approximation is in the same spirit as other fluid approximation literature in operations research (Avram et al. 1995, Gallego and van Ryzin 1994), our particular formulation incorporates elements of a linear dynamical system which we believe may be of independent interest in other DRA problems. For the queueing network control problem, we apply an existing optimization approach from the literature (Avram et al. 1995) that is based on a fluid approximation of the queueing network.

3. Through extensive computational experiments in both problems, we show the following:

(a) The MCTS and RHO approaches both produce high-quality solutions, generally performing as well or better than our customized heuristic for this problem. MCTS and RHO perform comparably when the problem instance is small. With a fixed computational budget, however, the RHO approach begins to outperform the MCTS approach as the size of the problem instance grows, either in state space or action space. Indeed, in the tactical wildfire management problem, MCTS can begin to perform worse than our baseline heuristic when the action space grows very large; the RHO approach, by comparison, still performs quite well. Similarly, for queueing network control, MCTS with an informed rollout policy (namely the well-known $c\mu$ rule in queueing theory) often performs worse than the same rollout policy on its own for larger queueing systems.

(b) The choice of hyperparameters in the MCTS algorithm—such as the exploration bonus and the progressive widening parameters—can significantly affect overall performance of the algorithm. The interdependence between these parameters is complex, and they cannot, in general, be selected independently. Some care must be taken to appropriately tune the algorithm to a specific DRA problem.

With regard to prior work in tactical wildfire management, there have been a number of empirically validated, deterministic models for wildfire spread proposed (e.g., Finney 2004, Tymstra et al. 2010). However, there have been fewer works that incorporate the stochastic elements of fire spread (Boychuck et al. 2008, Ntaimo et al. 2012, Fried et al. 2006). Most works focus on developing

models for the rate and simulation of the spread of wildfire; fewer consider the associated problem of managing suppression resources. A notable exception is the research stream of Hu and Ntaimo (2009) and Ntaimo et al. (2013), which proposes 1) an optimization formulation of the *initial attack* problem, i.e., the problem of determining how many and what kind of suppression resources to allocate to an ongoing fire and 2) a simulation engine to study how specific, user-specified heuristic suppression rules affect the growth of a fire. To the best of our knowledge, the authors do not address the tactical problem of *optimally* controlling suppression resources as the fire evolves. In other words, the authors do not consider the underlying DRA.

With regard to prior work in queueing network control, a variety of approaches have been proposed. These approaches include those based on heavy traffic approximations of the queueing network (Harrison 1988, Harrison and Wein 1990), fluid approximations (Avram et al. 1995, Meyn 2003, Paschalidis et al. 2004), characterizing the region of achievable performance (Bertsimas et al. 1994, 1995) and the linear programming approach to approximate dynamic programming (de Farias and Van Roy 2003, Bhat et al. 2012). To the best of our knowledge, MCTS has not been previously applied to problems in this domain.

The rest of this paper is organized as follows. In Section 2, we introduce our MDP formulation of the tactical wildfire management and the queueing network control problems that we use to compare the MCTS and RHO approaches. In Sections 3 and 4, we describe the MCTS and RHO approaches, respectively. In Sections 5 and 6, we describe our computational experiments for the wildfire and queueing control problems, respectively, and report on the results of these experiments. Finally, in Section 7, we summarize the main contributions of the work and highlight promising directions for future research.

2. Problem definition

We begin by describing the dynamics of the two problems that we will use to compare MCTS and RHO.

2.1. Tactical wildfire management

The first problem the we study is inspired by tactical wildfire management. In what follows, we present a model of tactical wildfire management that is simple enough to allow for a meaningful comparison of MCTS and RHO, while still capturing the key features of wildfire propagation, e.g., stochastic evolution, wind and topography effects and dependence on fuel. Although we do not explore the ideas here, it would be (at least conceptually) straightforward to extend our approach to a higher fidelity model incorporating moisture content and surface fuel-type as in Rothermel (1972) by using techniques similar to those developed in Hu et al. (2012).

Specifically inspired by the stochastic model of Boychuck et al. (2008) for wildland fire simulation, we model tactical wildland fire management as a Markov decision process. We partition the landscape into a grid of cells $\mathcal{X}$. There are two attributes for each cell $x \in \mathcal{X}$:

- $B(x)$, a Boolean variable indicating whether the cell is currently burning, and
- $F(x)$, an integer variable indicating how much fuel is remaining in the cell.

The collection of these two attributes over all cells in the grid represents the state of the Markov decision process.

We further assume a set of suppression resources $\mathcal{I}$ fighting the wildland fire. To simplify the model, we will treat all suppression resources as identical, but it is straightforward to extend to the case heterogenous resources with different capabilities as in Martin-Fernndez et al. (2002), Hu et al. (2012). The decisions of our MDP correspond to assigning resources to cells. For each $i \in \mathcal{I}$, let $a^{(i)} \in \mathcal{X}$ denote the cell to which we assign suppression resource i. We assume that any resource can be assigned to any cell at any time step, i.e., that the travel time between cells is negligibly small compared to the decision interval of the MDP.

Once ignited, a cell consumes fuel at a constant rate. Once the fuel is exhausted, the cell extinguishes. Since fuel consumption occurs at a constant rate, without loss of generality, we can rescale the units of time to make this rate equal to unity. Thus, we model the evolution of fuel in the model by

$$F_{t+1}(x) = \begin{cases} F_t(x) & \text{if } \neg B_t(x) \vee F_t(x) = 0 \\ F_t(x) - 1 & \text{otherwise.} \end{cases} \tag{1}$$

Notice this evolution is deterministic given $B_t(x)$.

The evolution of $B_t(x)$, however, is stochastic. Figure 1 shows the probabilistic transition model for $B_t(x)$ where

$$\rho_1 = \begin{cases} 1 - \prod_y (1 - P(x,y)B_t(y)) & \text{if } F_t(x) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and

$$\rho_2 = \begin{cases} 1 & \text{if } F_t(x) = 0 \\ 1 - \prod_i (1 - S(x)\delta_x(a^{(i)})) & \text{otherwise.} \end{cases} \tag{3}$$

Here, $P(x,y)$ is the probability that a fire in cell y ignites a fire in cell x. Generally, only the neighbors of x can ignite x, and so we expect $P(x,y)$ to be sparse. The specification of $P(x,y)$ can capture the tendency of a fire to propagate primarily in one direction due to wind or sloping terrain. $S(x)$ is the probability that a suppression effort on cell x successfully extinguishes the cell, while $\delta_x(a^{(i)})$ is an expression that is 1 if $a^{(i)} = x$ (suppression resource i is allocated to cell x) and 0 otherwise. We assume that the probability of success for multiple attempts on the same cell are independent.

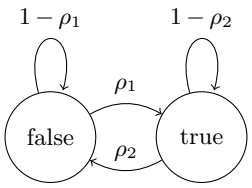


Figure 1 $B(x)$ **transition model.**

We stress that under these dynamics, cells that have been previously extinguished by a suppression team may later reignite.

The reward for a cell burning is $R(x)$ (always negative) and the total reward received at the tth step is $\sum_x B_t(x)R(x)$. We can vary the reward across the grid to represent a higher cost of a fire in particular areas. For example, we may penalize a fire in a populated area more heavily.

2.2. Control of queueing networks

The second problem that we study is the problem of queueing network control. In this problem, there is a network of servers and each server is fed by one or more queues/buffers that correspond to different job classes. Each job class is served by only one server, but a server may serve several different job classes. When a server becomes free, it can choose to process an available job from one of its classes. Once the server finishes, the job either exits the system, or proceeds to another server; at the new server, it becomes a job of a different class and waits in the queue for the class until the new server begins processing it. The time each server takes to process a job is random, with the distribution determined by the job's class. Jobs of some classes may arrive exogenously to the system. For the problem we will study, each such arrival process is an independent Poisson process, and the service time for each class is exponential.

As an example, let us consider the example network in Figure 2 from Bertsimas et al. (2015). In this example, there are two servers and three job classes. Jobs of class 1 and 2 arrive exogenously to the system with rates λ_1 and λ_2 respectively, and are serviced by server S_1 with rates μ_1 and μ_2 respectively. When a job of class 2 is completed, it exits the system. When a job of class 1 is completed, it becomes a job of class 3, and enters the corresponding buffer, where it awaits service from server S_2. The service time of a class 3 job in S_2 is exponential with rate μ_3; once a class 3 job completes service at S_2, it exits the system.

The queue sizes for the different job classes fluctuate over time as jobs arrive, enter service and are routed through the system. Whenever a server completes a job and becomes free, we must decide which queue (i.e., which job class) the server will draw its next job from. The problem of queueing network control is to decide at each such decision epoch which job class will be served so as to minimize the expected weighted long-run average number of jobs in the system.

Formally, a queueing network control problem is described by the following data:

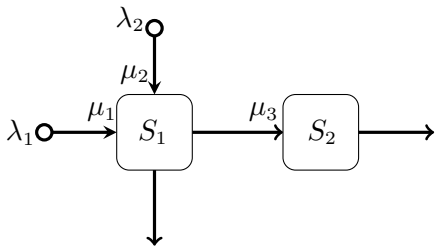


Figure 2 Criss-cross network from Bertsimas et al. (2015).

- n: the number of job classes.
- m: the number of servers.
- $s(i)$: server of class i.
- λ_i: rate of exogenous Poisson arrival process to class i. (Note that λ_i may be 0, in which case there are no exogenous arrivals to class i.)
- μ_i: exponential service time rate for job class i when it is served by server $s(i)$.
- g_{ij}: routing indicator; $g_{ij} = 1$ if class i jobs become class j jobs after service at server $s(i)$, and 0 otherwise. Note that the case $g_{ij} = 0$ for every j denotes the fact that jobs of class i, once processed by $s(i)$, exit the system, and do not join a queue at a different server.
- $\mathbf{c} = (c_1, \ldots, c_n)$: vector of weights for each class (used to determine the objective).
- T: the time limit for the queueing problem.

We let $\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$ denote the number of jobs of class i in the system (including a job that may be in service) at time t. The objective is to minimize the expected weighted long-run average number of jobs in the system from $t = 0$ to $t = T$, which is given by the following performance metric:

$$\frac{1}{T} \cdot \mathbb{E}_\pi \left[\int_0^T \mathbf{c}^T \mathbf{x}(t)\, dt \right],$$

where π denotes the policy used to decide which class to serve at each decision epoch.

3. A Rolling Horizon Optimization Approach

We now present rolling horizon optimization approaches for the tactical wildfire management problem and for the queueing network control problem.

3.1. Approach for tactical wildfire management

In this section, we present an optimization-based solution approach for the tactical wildfire management problem. In this approach, we formulate a deterministic optimization problem that approximates the original MDP. At each decision step, we re-solve this approximation based on the current state of the process and select the first prescribed allocation.

The key feature of the formulation is the use of a deterministic, "smoothed" version of the dynamics presented in Section 2. Rather than modeling the state with a discrete level of fuel and

binary state of burning, we model fuel as a continuous quantity and model a new (continuous) *intensity* level of each cell representing the rate at which fuel is consumed. Then, rather than representing the evolution of cells being burning/not burning using a probabilistic transition model (cf. Figure 1), we model the evolution of intensity through one-step linear dynamics. Other authors have used similar ideas when motivating various fluid approximations in the operations research literature. For example, as we will shortly discuss in Section 3.2, continuous fluid approximations have been used to study the control of queuing networks, where the size of each queue in the network can be modeled continuously through systems of differential equations, and the decision at each time for each server in the network is the "rate" at which each queue is being served/emptied. Another example comes from revenue management (RM). In a typical RM problem, the decision maker needs to sell some fixed inventory of a product, and must at each point in time set a price for this product. The price determines the rate of a stochastic demand process, with the goal of maximizing the total revenue at the end of the selling period. The optimal solution of the exact problem is, with few exceptions, extremely difficult to obtain. However, if the dynamics of the problem are relaxed so that the demand is continuous and arrives at a deterministic rate that varies with the price, one can obtain useful bounds and near-optimal pricing policies for the exact, *stochastic* problem Gallego and van Ryzin (1994).

As we will see when we present the formulation below, smoothing the dynamics in this way results in a tractable optimization model that can be solved to obtain an allocation at each time step. We wish to emphasize that the formulation that we solve at each period is *not* identical to the original problem. As a result, the allocation derived from our formulation at a given period is not guaranteed to be the same as the optimal allocation that would result from solving the original MDP in Section 2.1. Nevertheless, given the wide application of fluid models to other problems in the operations research literature that are similarly intractable (such as the queueing control and revenue management examples highlighted above), it seems reasonable to expect that our approach here may still yield good policies for the original MDP in Section 2.1.

3.1.1. Optimization Model Let $A_t(x,i)$ be a binary variable that is 1 if suppression resource $i \in \mathcal{I}$ is assigned to cell x at time t and 0 otherwise; this is the main decision variable of the problem. Recall that $F_t(x)$ denotes the amount of fuel available at the start of period t in cell x. Furthermore, let $I_t(x)$ represent the intensity of the fire in cell $x \in \mathcal{X}$ at time t. Intensity is a continuous, nonnegative decision variable that will be determined by the optimization algorithm. Unlike the original MDP formulation, it is no longer possible to rescale the parameters so that exactly one unit of fuel is consumed per period. (We discuss how to calibrate the fuel appropriately next.)

Some additional notation is required to describe the evolution of $I_t(x)$. Define $\mathcal{N}(x)$ as the set of cells that are neighbors of x in the sense of fire transmission, i.e. $\mathcal{N}(x) = \{y : P(x,y) > 0\}$. Let $\zeta_t(y,x) \in [0,1]$ be the rate at which the intensity $I_{t-1}(y)$ of cell y at time $t-1$ contributes to the intensity $I_t(x)$ of cell x at time t. Furthermore, let $\tilde{\zeta}_t(x,i) \in [0,1]$ be the relative reduction in intensity when suppression team i is assigned to cell x at time t. Finally, let $I_0(x) = 1$ if cell x is burning at time 0, and $I_0(x) = 0$ if cell x is not burning. Note that these values are simply the initial values of the intensity variables and $I_t(x)$ for $t > 0$ can take on any nonnegative value.

With this notation, our formulation is

$$\text{minimize} \quad \sum_{x\in\mathcal{X}} \sum_{t=0}^{T} R(x) I_t(x) \tag{4a}$$

$$\begin{aligned}\text{subject to} \quad & I_t(x) \geq I_{t-1}(x) + \sum_{y\in\mathcal{N}(x)} I_{t-1}(y)\cdot\zeta_t(y,x) \\ & - \sum_{i\in\mathcal{I}} A_{t-1}(x,i)\cdot\bar{I}_t(x)\cdot\tilde{\zeta}_t(x,i) - \left(F_0(x) + \sum_{y\in\mathcal{N}(x)} F_0(y)\right)\cdot z_{t-1}(x), \\ & \quad \forall x\in\mathcal{X},\ t\in\{1,\ldots,T\},\end{aligned} \tag{4b}$$

$$F_t(x) = F_0(x) - \sum_{t'=0}^{t-1} I_{t'}(x), \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4c}$$

$$F_t(x) \geq \delta\cdot(1-z_t(x)), \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4d}$$

$$F_t(x) \leq \delta\cdot z_t(x) + F_0(x)\cdot(1-z_t(x)), \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4e}$$

$$I_{t+1}(x) \leq F_0(x)\cdot(1-z_t(x)), \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4f}$$

$$\sum_{x\in\mathcal{X}} A_t(x,i) \leq 1, \quad \forall t\in\{0,\ldots,T\},\ i\in\mathcal{I}, \tag{4g}$$

$$I_t(x),\ F_t(x) \geq 0, \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4h}$$

$$z_t(x)\in\{0,1\}, \quad \forall x\in\mathcal{X},\ t\in\{0,\ldots,T\}, \tag{4i}$$

$$A_t(x,i)\in\{0,1\}, \quad \forall x\in\mathcal{X},\ i\in\mathcal{I},\ t\in\{0,\ldots,T\}. \tag{4j}$$

Here, $\delta > 0$ is a small threshold chosen so that a cell with less than δ units of fuel cannot burn. Consequently, the binary decision variable $z_t(x)$ represents the indicator that the fuel at time t in cell x is below δ. Finally in the spirit of "big-M" constraints, $\bar{I}_t(x)$ is an upperbound on the maximal value attainable by $I_t(x)$. We will discuss how to compute this value shortly.

The constraints have the following meaning:

• Constraint (4b) expresses the one-step dynamics of the fire intensity in region x at time t. Although we have written the constraint in inequality form, it is not difficult to see that in an

optimal solution, the constraint will always be satisfied at equality since the objective is a sum of the intensities over all periods and regions weighted by the (positive) importance factors.

The first two terms of the right-hand side represent that—without intervention and without regard for fuel—the intensity of a cell one step into the future is the current intensity ($I_{t-1}(x)$) plus the sum of the intensities of the neighboring cells weighted by the transmission rates ($\sum_{y\in\mathcal{N}(x)} I_{t-1}(y)\cdot\zeta_t(y,x)$). If suppression team i is assigned to cell x at $t-1$, then $A_t(x,i)=1$ and the intensity is reduced by $\tilde{\zeta}_t(x,i)\cdot\bar{I}_t(x)$. If the cell's fuel is below δ at time $t-1$, then $z_t(x)=1$, and the intensity is reduced by $F_0(x)+\sum_{y\in\mathcal{N}(x)} F_0(y)$; since the intensity of a cell is upper bounded by the initial fuel of that cell, the term $-\left(F_0(x)+\sum_{y\in\mathcal{N}(x)} F_0(y)\right)\cdot z_{t-1}(x)$ ensures that whenever the fuel $F_t(x)$ drops below δ, this constraint becomes vacuous.

- Constraint (4c) is the equation for the remaining fuel at a particular time point as a function of the intensities (intensity is assumed to be the fuel burned in a particular time period).
- Constraint (4d) and (4e) are forcing constraints that force $F_t(x)$ to be between δ and $F_0(x)$ if $z(t)=0$, and between 0 and δ if $z(t)=1$.
- Constraint (4f) ensures that if there is insufficient fuel in cell x at period t, then the intensity at that cell in the next time point is zero. If there is sufficient fuel, then the constraint is vacuous (the intensity is at most $F_0(x)$, which is already implied in the formulation).
- Constraint (4g) ensures that each team in each period is assigned to at most one cell.
- The remaining constraints ensure that the fuel and intensity are continuous nonnegative variables, and that the sufficient fuel and team assignment variables are binary.

The objective (4a) is the sum of the intensities over all of the time periods and over all cells, weighted by the importance factor of each cell in each time period.

Problem (4) is a mixed integer linear optimization model with two sets of binary variables: the $A_t(x,i)$ variables, which model the assignment of suppression teams to cells over time, and the $z_t(x)$ variables, which model the loss of fuel over time. Although mixed linear optimization is not solvable in polynomial time, there exist high-quality open-source and commercial solvers which are able to solve such problems extremely efficiently in practice, even for very large instances.

In highly resource constrained environments when it is not possible to solve the above model to optimality, we can still obtain extremely good approximate solutions by relaxing the $A_t(x,i)$ variables to be continuous within the unit interval $[0,1]$. Then, given an optimal solution with fractional values for the $A_t(x,i)$ variables at $t=0$, we can compute a score $v(x)$ for each cell x as $v(x)=\sum_{i\in\mathcal{I}} A_0(x,i)$. We then assign suppression teams to the $|\mathcal{I}|$ cells with the highest values of the index v. Indeed, we will follow this strategy in Section 5.

3.1.2. Calibrating the Model Given parameters for the original MDP formulation of the tactical wildland fire management problem, parameters for our nominal optimization formulation are obtained as follows:

- $\bar{I}_t(x)$ is computed by iterating a modified version of the one-step recursion, assuming that there is no intervention and infinite fuel:

$$\bar{I}_t(x) = \bar{I}_{t-1}(x) + \sum_{y \in \mathcal{N}(x)} \bar{I}_{t-1}(y),$$

where $\bar{I}_0(x) = I_0(x)$. In this modified one-step recursion, each transmission rate $\zeta_t(y,x)$ is essentially assumed to be 1, which is the highest value it can be.

- $F_0(x)$ is obtained by summing the fuel threshold δ and the $\bar{I}_t(x)$ values over the horizon $t = 0, 1, \ldots, \min\{T, F(x)\}$, where $F(x)$ is the number of periods that cell x can burn into the future according to the original MDP dynamics:

$$F_0(x) = \delta + \sum_{t=0}^{\min\{T,F(x)\}} \bar{I}_t(x).$$

Intuitively, since the intensity $I_t(x)$ can be thought of as how much fuel was consumed by the fire in cell x at time t, the initial fuel value $F_0(x)$ can be thought of as a limit on the cumulative intensity in a cell over the entire horizon. Once the cumulative intensity has reached $\sum_{t=0}^{\min\{T,F(x)\}} \bar{I}_t(x)$, the fuel in the cell enters the interval $[0, \delta]$, at which point the variable $z_t(x)$ is forced to 1 and the intensity is forced to zero for all remaining time periods.

- $\zeta_t(y,x)$ is set to $P(x,y)$ (the transmission probability from y to x) for each t.
- $\tilde{\zeta}_t(x,i)$ is set to $Q(x)$ (the probability of successful extinguishing a fire in cell x) for each period t and each suppression team i.
- δ is set to 0.1.

3.2. Approach for queueing network control

In this section, we describe the optimization-based approach that we will use for the problem of queueing network control. In particular, we will apply the fluid optimization technique first developed in Avram et al. (1995) and later extended in Bertsimas et al. (2015). For completeness, we will describe the model here. In this approach, the key idea is to relax the discrete stochastic dynamics of the original system and replace them with continuous, deterministic dynamics. In the tactical wildfire problem of Section 3.1, this was accomplished by replacing probabilistic transition dynamics with one-step linear dynamics that relate the intensities of the cells at the current period to the intensities of the cell at the next period. For the queueing network control problem that we now focus on, we will instead replace the probabilistic arrival and service dynamics with a system

of deterministic differential equations. Again, we emphasize that the new optimization problem that we solve here is not equivalent to the true MDP in Section 2.2 and as such, the resulting decision of which queue each server should work on is not necessarily optimal for the true MDP. However, it is well-known that fluid approximations of queueing networks are closely related to the true queueing network. For example, the true queueing network is stable if its associated fluid approximation is stable (Dai 1995). Moreover, simulation results of fluid-based approaches have shown them to perform very well (Avram et al. 1995, Bertsimas et al. 2015).

3.2.1. Optimization Model The decision variables are $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ for $t \in [0, T]$, where $x_i(t)$ indicates the *fluid* of class i at time t, and $\mathbf{u}(t) = (u_1(t), \ldots, u_n(t))$ where $u_i(t)$ denotes the effort (rate of service) that is devoted to queue i by its server $s(i)$.

We use the same data in Section 2.2. For convenience, we define the matrix $\mathbf{A}$ as

$$a_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if class } i \text{ jobs become class } j \text{ jobs upon service by server } s(i), \\ 0 & \text{otherwise.} \end{cases}$$

The dynamics of the fluid of class i can be written as

$$\dot{x}_i(t) = \lambda_i - u_i(t) - \sum_{j \neq i} a_{ji} u_j(t); \tag{5}$$

in words, the rate of change of the fluid of class i at time t is the exogenous arrival rate λ_i, less the rate at which the class is being serviced $u_i(t)$, plus the rate at which other classes are serviced that become jobs of this class (the $-\sum_{j \neq i} a_{ji} u_j(t)$ term). In matrix form, this constraint becomes

$$\dot{\mathbf{x}}(t) = \boldsymbol{\lambda} - \mathbf{A}\mathbf{u}(t), \quad \forall t \in [0, T]. \tag{6}$$

By integrating on both sides for every t, the constraint can be re-written as

$$\int_0^t \mathbf{A} u(s)\, ds + \mathbf{x}(t) = \mathbf{x}(0) + \boldsymbol{\lambda} t, \quad \forall t \in [0, T] \tag{7}$$

where $\mathbf{x}(0)$ is a vector of the number of jobs of each class in the system at time 0.

The variable $u_i(t)$ denotes the rate at which class i is being serviced by server $s(i)$. These rates are constrained in the following way: for a given server s, we must have

$$\sum_{i:s(i)=s} \frac{u_i(t)}{\mu_i} \leq 1, \quad \forall t \in [0, T]. \tag{8}$$

In words, $u_i(t)/\mu_i$ represents the rate that class i is being serviced as a fraction of server s's (maximum) rate for class i, and so the constraint ensures that in a fluid sense, the server is serving at most one job per time unit. By defining the matrix $\mathbf{H}$ component-wise as

$$h_{si} = \begin{cases} 1/\mu_i & \text{if } s(i) = s, \\ 0 & \text{otherwise,} \end{cases}$$

the constraint can be written in matrix form as

$$\mathbf{Hu}(t) \leq \mathbf{1}, \quad \forall t \in [0,T]. \tag{9}$$

where $\mathbf{1}$ is a column vector of m ones.

With these definitions, the fluid optimization problem can be written as

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{T}\int_0^T \mathbf{c}^T\mathbf{x}(t)\,dt && \text{(10a)}\\
\text{subject to} \quad & \int_0^t \mathbf{A}u(s)\,ds + \mathbf{x}(t) = \mathbf{x}(0) + \boldsymbol{\lambda} t, \quad \forall t \in [0,T], && \text{(10b)}\\
& \mathbf{Hu}(t) \leq \mathbf{1}, \quad \forall t \in [0,T], && \text{(10c)}\\
& \mathbf{u}(t), \mathbf{x}(t) \geq 0, \quad \forall t \in [0,T], && \text{(10d)}
\end{aligned}$$

where the objective is the long-run average weighted fluid in the system and the last two constraints require that the effort devoted to each class and the fluid of each class are never below zero.

The problem can be further re-formulated to eliminate the $\mathbf{x}(t)$ variable, leaving only the control variable $\mathbf{u}(t)$:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{T}\int_0^T (T-t)\mathbf{c}^T(\boldsymbol{\lambda} - \mathbf{Au}(t))\,dt && \text{(11a)}\\
\text{subject to} \quad & \int_0^t \mathbf{A}u(s)\,ds \leq \mathbf{x}(0) + \boldsymbol{\lambda} t, \quad \forall t \in [0,T], && \text{(11b)}\\
& \mathbf{Hu}(t) \leq \mathbf{1}, \quad \forall t \in [0,T], && \text{(11c)}\\
& \mathbf{u}(t) \geq 0, \quad \forall t \in [0,T]. && \text{(11d)}
\end{aligned}$$

This problem is a semi-continuous linear programming (SCLP) problem that has been studied quite extensively (see, e.g., Pullan 1993, 1995, 1996, Fleischer and Sethuraman 2005). Although the problem appears to be intractable (having an infinite number of variables and constraints), it turns out that it is possible to characterize the structure of the optimal solution (namely, the control $\mathbf{u}(t)$ is a piecewise constant function) and there exist efficient numerical algorithms for solving this problem that exploit this (e.g., Luo and Bertsimas 1998). Mathematically, the optimal control $\mathbf{u}^*(t)$ takes the form

$$\mathbf{u}(t) = \left\{ \mathbf{u}_k,\ t_k \leq t < t_{k+1}, \right.$$

where $\{t_0, t_1, \ldots, t_q\}$ is a partition of the interval $[0,T]$ ($t_0 = 0$, $t_q = T$) and $\mathbf{u}_k$ is the control value on the interval $[t_k, t_{k+1})$.

The fluid control approach to the queueing network control problem is, at each decision epoch, to do as follows:

1. Set $\mathbf{x}(0)$ to reflect the current number of jobs of each class in the system.
2. Solve problem (11) to obtain the optimal control $\mathbf{u}^*(t)$.
3. At each server s that is idle, serve the non-empty class i of that server with the highest value of $u_i^*(0)$.

4. Monte Carlo Tree Search

Our review of MCTS is necessarily brief; the reader is referred to Browne et al. (2012) for a more thorough survey. In particular, we will focus on a variation of MCTS that employs *double progressive widening*, a technique that explicitly controls the branching factor of the search tree (Couëtoux et al. 2011). This variation is specifically necessary when the action space is continuous or so large that all actions cannot possibly be explored. This is frequently the case in most DRA problems. We note that there exist other strategies for managing the branching factor, such as the pruning strategy employed in the "bandit algorithm for smooth trees" (BAST) method of Coquelin and Munos (2007b); we do not consider these here.

We first describe MCTS with double progressive widening in general, and then discuss two particular modifications we have made to the algorithm to tailor it to DRAs.

4.1. MCTS with double progressive widening

Algorithm 1 involves running many simulations from the current state while updating an estimate of the state-action value function $Q(s,a)$. Each simulation from the current state is executed to a depth of d. We use a generative model G to produce samples of the next state s' and reward r given the current state s and action a. We draw samples $(s',r) \sim G(s,a)$. All of the information about the state transitions and rewards is represented by G; the state transition probabilities and expected reward function are not used directly. There are three stages in each simulation: search, expansion, and rollout.

4.1.1. Search If the current state in the simulation is in the set T (initially empty), we enter the search stage. Otherwise, we proceed to the expansion stage. During the search stage, we update $Q(s,a)$ for the states and actions visited and tried in our search. We also keep track of the number of times we have visited a state $N(s)$ and the number of times we have taken an action from a state $N(s,a)$.

During the search, the first progressive widening controls the number of actions considered from a state. To do this, we generate a new action if $\|A(s)\| < kN(s)^\alpha$, where k and α are parameters that control the number of actions considered from the current state and $A(s)$ is the set of actions tried from s. When generating a new action, we add it to the set $A(s)$, and initialize $N(s,a)$ and $Q(s,a)$ with $N_0(s,a)$ and $Q_0(s,a)$, respectively. The functions N_0 and Q_0 can be based on prior expert knowledge of the problem; if none is available, then they can both be initialized to 0. We also initialize the empty set $V(s,a)$, which contains the set of states s' transitioned to from s when selecting action a. A default strategy for generating new actions is to randomly sample from candidate actions. After potentially generating new actions, we execute the action that maximizes

$$Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}},$$

Algorithm 1 Monte Carlo tree search with double progressive widening

function MonteCarloTreeSearch(s, d)
  loop
    Simulate(s, d)
  return $\arg\max_a Q(s, a)$
function Simulate(s, d)
  if $d = 0$ **then**
    return 0
  if $s \notin T$ **then**
    $T = T \cup \{s\}$
    $N(s) \leftarrow N_0(s)$
    return Rollout(s, d)
  $N(s) \leftarrow N(s) + 1$
  if $\|A(s)\| < kN(s)^{\alpha}$ **then**
    $a \leftarrow$ Getnext(s, Q)
    $(N(s, a), Q(s, a), V(s, a)) \leftarrow (N_0(s, a), Q_0(s, a), V(s, a))$
    $A(s) = A(s) \cup \{a\}$
  $a \leftarrow \arg\max_a Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$
  if $\|V(s, a)\| < k'N(s, a)^{\alpha'}$ **then**
    $(s', r) \sim G(s, a)$
    if $s' \notin V(s, a)$ **then**
      $V(s, a) = V(s, a) \cup \{s'\}$
      $N(s, a, s') \leftarrow N_0(s, a, s')$
    else
      $N(s, a, s') \leftarrow N(s, a, s') + 1$
  else
    $s' \leftarrow$ Sample$(N(s, a, \cdot))$
    $N(s, a, s') \leftarrow N(s, a, s') + 1$
  $q \leftarrow R(s, a, s') + \gamma$Simulate$(s', d - 1)$
  $N(s, a) \leftarrow N(s, a) + 1$
  $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q
function Rollout(s, d)
  if $d = 0$ **then**
    return 0
  $a \sim \pi_0(s)$
  $(s', r) \sim G(s, a)$
  return $r + \gamma$Rollout$(s', a, d - 1)$

where c is a parameter that controls the amount of exploration in the search. The second term is an *exploration bonus* that encourages selecting actions that have not been tried as frequently.

Next, we draw a sample $(s', r) \sim G(s, a)$, if $\|V(s, a)\| < k' N(s, a)^{\alpha'}$. In this second progressive widening step, the parameters k' and α' control the number of states transitioned to from s. If s' is not a member of $V(s, a)$, we add it to the set $V(s, a)$, initialize $R(s, a, s')$ to r, and initialize $N(s, a, s')$ with $N_0(s, a, s')$. If s' is a member of $V(s, a)$, then we increment $N(s, a, s')$. However, if $\|V(s, a)\| \geq k' N(s, a)^{\alpha'}$, then we select s' from $V(s, a)$ proportional to $N(s, a, s')$.

4.1.2. Expansion Once we have reached a state that is not in the set T, we initialize $N(s, a)$ with $N_0(s, a)$, add the current state to the set T and proceed to the rollout stage.

4.1.3. Rollout After the expansion stage, we simply select actions according to some *rollout* (or default) policy π_0 until the desired depth is reached. Typically, rollout policies are stochastic, and so the action to execute is sampled $a \sim \pi_0(s)$. The rollout policy does not have to be close to optimal, but it is a way for an expert to bias the search into areas that are promising. The expected value is returned and is used in the search to update the value for $Q(s, a)$ used by the search phase.

Simulations are run until some stopping criteria is met, often simply a fixed number of iterations. We then execute the action that maximizes $Q(s, a)$. Once that action has been executed, we can rerun Monte Carlo tree search to select the next action. It is common to carry over the values of $N(s, a)$, $N(s)$, and $Q(s, a)$ computed in the previous step.

4.2. Tailoring MCTS to DRAs

We next discuss two modifications we have found to be critical when applying MCTS to DRAs.

4.2.1. Action Generation As previously mentioned, the default strategy for generating new actions during the search stage of MCTS involves randomly sampling an action from all candidate actions. In DRAs where the action space may be very large, this strategy is inefficient; we may need to search many actions before identifying a high-quality choice. Rather, we would like to bias the sampling towards potential actions that we believe may perform well. One way to identify such actions is via some application specific heuristic.

We follow a slightly different approach. Specifically, consider MCTS after several iterations. The current values of $Q(s, a)$ provide a (noisy) estimate of the value function, and, hence, can be used to approximately identify promising actions. Consequently, we use these estimates to bias our sampling procedure through a sampling scheme inspired by genetic algorithm search heuristics (Whitley 1994). Our strategy, described in Algorithm 2, involves generating actions using one of three approaches: with probability u' an existing action in the search tree is mutated, with

probability u'' two existing actions in the search tree are recombined, or a new action is generated from the default strategy. Mutating involves randomly changing the allocation of one or resources. Recombining involves selecting a subset of allocations from two actions and combining the two subsets. When mutating or recombining, we select the existing action (or actions) from $A(s)$ using tournament select where the fitness for each action is proportional to $Q(s,a)$. Note that it is permissible to use other methods such as softmax, too.

Algorithm 2 Action Generation

function GETNEXT(s, Q)
  $u \sim U(0,1)$
  if $u < u'$ **then**
    $a' \leftarrow$ SAMPLE($Q(s,\cdot)$)
    $a \leftarrow$ MUTATE(a')
  else if $u < u' + u''$ **then**
    $a' \leftarrow$ SAMPLE($Q(s,\cdot)$)
    $a'' \leftarrow$ SAMPLE($Q(s,\cdot)$)
    $a \leftarrow$ RECOMBINE(a', a'')
  else
    $a \sim \pi_0(s)$
  return a

Our numerical experiments confirm that our proposed action generation approach significantly outperform the default strategies in our benchmark problem.

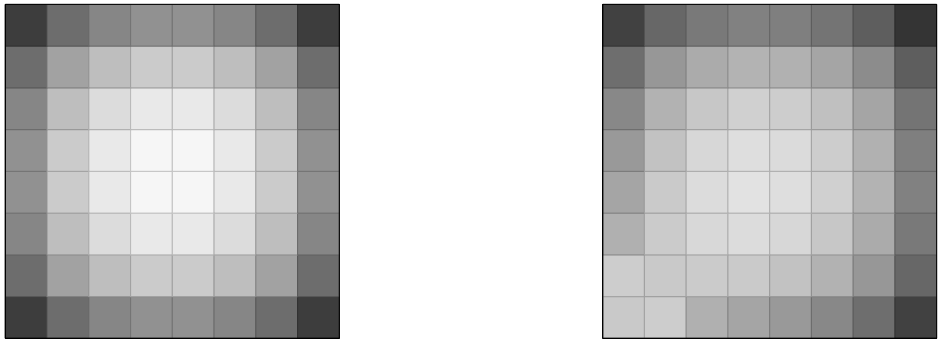
4.2.2. Rollout Policy In many papers treating MCTS, it is argued that even if the heuristic used for the rollout policy is highly suboptimal, given enough time the algorithm will converge to the correct state-action value function $Q(s,a)$. In DRAs with combinatorial structure (and, hence, huge state and action spaces), it may take an extremely long time for this convergence. (Similar behavior was observed in Coquelin and Munos 2007a). Indeed, we have observed for DRAs that having a good initial rollout policy makes a material difference in the performance of MCTS. Unfortunately, designing a good heuristic seems to be an application specific task.

For the tactical wildfire management problem, we consider a heuristic that involves assigning a weight to each cell x

$$W(x) = \sum_y \frac{R(y)}{D(x,y)}, \tag{12}$$

where $D(x,y)$ is the shortest path between x and y assuming that the distance between adjacent cells is $P(x,y)$. We compute the values $D(x,y)$ offline using a graph analysis algorithm, such as the

Floyd-Warshall algorithm (Floyd 1962), and consequently term this heuristic the FW heuristic. Figure 3 shows example weights assigned to different cells on an eight by eight grid with two different reward profiles. The heuristic performs generally well because it prioritizes allocating resources to cells that are near large negative reward cells (i.e., populated areas). The rollout



(a) Uniform reward. (b) Large negative reward in lower left.

Figure 3 Example heuristic weights. Lighter cells correspond to higher weights.

policy involves selecting the cells that are burning and assigning resources to the highest weighted cells. We are also able to randomly sample from the weights to generate candidate actions. In our experiments, we have observed that the heuristic performs fairly well, and, consequently, may be of independent interest to the wildfire suppression community.

For the queueing network control problem, in addition to a random policy, we also consider the so-called $c\mu$ heuristic. Under this heuristic, when a server becomes idle, the class it draws its next job from is the class i with the highest value of $c_i\mu_i$ (the service rate weighted by the cost of the class). This type of policy is simple to implement and for many special cases of queueing systems, it is known to be either optimal or asymptotically optimal (see, for example, Coffman Jr and Mitrani 1980, Buyukkoc et al. 1985, Van Mieghem 1995).

5. Numerical Comparisons for Tactical Wildfire Management

This section presents experiments comparing Monte Carlo tree search (MCTS) and the rolling horizon optimization (RHO) approach. We seek to understand their relative strengths and weaknesses as well as how the user-defined parameters of each approach, such as the exploration bonus c for MCTS, affect the performance of the algorithm. Before proceeding to the details of our experiments, we summarize our main insights here:

- Overall, RHO performs as well as or better than MCTS. For even moderate computational budgets, RHO generates high quality solutions.
- Although the MCTS approach works well for certain smaller examples, its performance can degrade for larger examples (with a fixed, computational budget). Moreover, the dependence on

the algorithm on its underlying hyperparameters is complex. The optimal choice of the exploration bonus and progressive widening factors depends both on the underlying heuristic used in the rollout as well as available computational budget.

5.1. Algorithmic Parameters and Basic Experimental Setup

In what follows, we use a custom implementation of MCTS written in C++ and use the mixed-integer optimization software Gurobi 5.0 (Gurobi Optimization, Inc. 2013) to solve the RHO formulation. All experiments were conducted on a computational grid with 2.2 GHz cores with 4GB of RAM in a single-threaded environment. Although it is possible to parallelize many of the computations for each of the four methods, we do not explore this possibility in these experiments.

Many of our experiments will study the effect of varying various hyperparameters (like the time limit per iteration) on the solution quality. Unless otherwise specified in the specific experiment, all hyperparameters are set to their baseline values in Table 1.

Table 1 Default parameters for algorithms

Method	Parameter	Value
MCTS	Time Limit per Iteration	60 s
	Exploration Bonus c	50
	Rollout Policy	FW Heuristic
	Depth d	10
	Progressive Widening, Action Space α	0.5
	Progressive Widening, State Space α'	0.2
	Progressive Widening, Action Space k	40
	Progressive Widening, State Space k'	40
	Algorithm 2 (u', u'')	(0.3, 0.3)
RHO	Time Limit per Iteration	60 s
	Horizon Length	10

To give a sense of the how many trajectories are generated by MCTS each time it is called, Table 2 shows the average over ten random grid configurations of the number of MCTS simulation iterations/trajectories for varying grid sizes k and varying numbers of teams a. The default parameters in Table 1 were used. Each grid configuration was generated according to the procedure for grid 1 (described in the next section). From this table, we can see that even in the largest instance, MCTS typically generates over 100,000 trajectories of the system.

To ease comparison in what follows, we generally present the performance of each our algorithms relative to the performance of a randomized suppression heuristic. At each time step, the randomized suppression heuristic chooses $|\mathcal{I}|$ cells without replacement from those cells which are currently burning and assigns suppression teams to them. This heuristic should be seen as a naive

Table 2 Average number of MCTS iterations/trajectories for MCTS with default parameters for grid one with varying grid size k and number of resources/teams a.

	Grid size k		
Num. teams a	8	16	20
4	445,450.4	213,801.7	169,753.2
8	314,777.4	172,282.6	127,112.7
12	298,593.7	146,611.8	102,701.0

"straw man" for comparisons only. We will also often include the performance of our more tailored heuristic, the Floyd-Warshall (FW) heuristic, as a more sophisticated straw man.

There are two experimental setups that we will return to throughout this section.

5.1.1. Grid 1 In this setup, we consider a $k \times k$ grid with a varying reward function. There is a negative one reward received when the lower left cell is burning and the reward for a cell burning increases by one when traversing up or to the right across the grid. Also, the reward in the upper right hand corner is always -10. Figure 4 shows the rewards for a $k = 8$ grid.

-8	-9	-10	-11	-12	-13	-14	-10
-7	-8	-9	-10	-11	-12	-13	-14
-6	-7	-8	-9	-10	-11	-12	-13
-5	-6	-7	-8	-9	-10	-11	-12
-4	-5	-6	-7	-8	-9	-10	-11
-3	-4	-5	-6	-7	-8	-9	-10
-2	-3	-4	-5	-6	-7	-8	-9
-1	-2	-3	-4	-5	-6	-7	-8

Figure 4 Rewards for Grid 1 with $k = 8$.

The fire in this experiment propagates as described in Section 2 with

$$P(x, y) = \begin{cases} 0.06 & \text{if } y \in \mathcal{N}(x) \\ 0 & \text{otherwise.} \end{cases}$$

We also assume for this experiment that suppression efforts are successful with an 80% probability—that is, $Q(x) = 0.8$ for all $x \in \mathcal{X}$.

For a single simulation we randomly generate an initial fire configuration—that is, whether or not each cell is burning and the fuel level in each cell. After generating an initial fire, suppression then begins according to one of our four approaches with $|\mathcal{I}|$ teams. The simulation and suppression efforts continue until the fire is extinguished or the entire area is burned out. A typical experiment

will repeat this simulation many times with different randomly generate initial fire configurations and aggregate the results.

Table 3 Grid 1 initial fire statistics

	$k=8$	$k=12$	$k=16$	$k=20$	$k=30$
Mean cells burning	37.6	91.4	168.7	275.5	664.2
Maximum cells burning	62	142	244	372	845
Mean fuel level for burning cells	15.8	19.9	22.8	25.7	31.4
Fuel level for non-burnt cells	24	29	34	38	46

The specific process for initializing the fire configuration in a simulation is as follows:

1. Initialize all of the cells with a fuel level of $\lfloor \frac{k}{2P(x,y)} \rfloor$ and seed a fire in the lower left hand cell.

2. Allow the fire to randomly propagate for $\lfloor \frac{k}{2P(x,y)} \rfloor$ steps. Note that the lower left hand cell will naturally extinguish at the point.

3. Next, scale the fuel levels by a factor of $k^{-0.25}$. We scale the fuel levels to reduce the length of experiments where the number of suppression teams is insufficient to successfully fight the fire. Table 3 shows summary statistics for the initial fire configurations that arise from this process for this experiment. We stress that the primary goal of this experiment is to explore the scalability of the various approaches.

5.1.2. Grid 2 This setup mirrors the setup in the previous experiment with two exceptions. First, we initialize fires in the middle of the grid. Second, the reward function for cells is exponential across the grid.

Specifically, at time $t=0$ we ignite the cell in the middle of the grid, i.e., the cell at location $(\lceil k/2 \rceil, \lceil k/2 \rceil)$. The reward for cell $x=(i,j)$ is $-C \cdot \exp(-\lambda i)$ where $C^{-1} \equiv \sum_{i=1}^{k} e^{-\lambda i}$. Notice that the reward only depends on the horizontal location of the cell in the grid. In other words, cells located to the left are more valuable. The value of λ controls the rate at which the reward grows. Some typical reward curves are shown in Figure 5 for a $k=20$ grid. Observe that for large values of λ, the local reward structure at the site of the fire may seem quite flat. Good policies need to account for the fact that despite this local structure, suppression of cells on the left-hand side of the fire is ultimately more valuable than suppression to the right.

In this experiment, suppression efforts are still 80% successful. The spread probabilities are given by

$$P(x,y) = \begin{cases} 0.02 & \text{if } y \in \mathcal{N}(x) \\ 0 & \text{otherwise.} \end{cases}$$

As in the previous experiment, we begin with a random initial fire configuration. The specific process for initializing fire situations in this experiment is as follows:

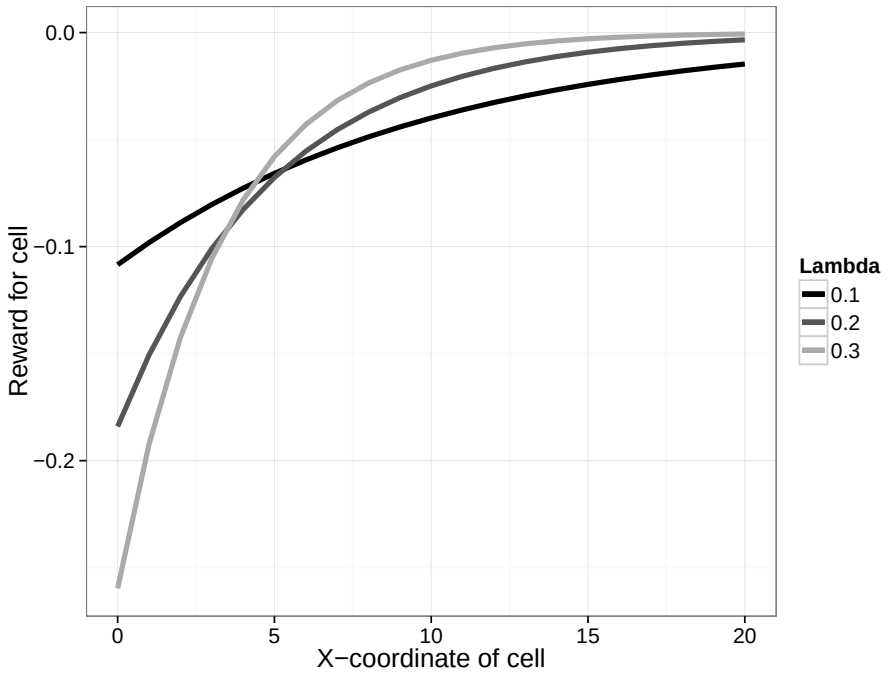


Figure 5 **Grid 2 reward structure for various values of λ.**

1. Initialize the cells with fuel levels of $\lfloor \frac{k}{4P(x,y)} \rfloor$ and seed the fire in the cell in the middle of the grid—that is, the cell at location $(\lceil k/2 \rceil, \lceil k/2 \rceil)$.

2. Allow the fire to randomly spread for $\lfloor \frac{k}{4P(x,y)} \rfloor$ steps. Note that we selected smaller fuel levels and initial times for the fire to build because the fire now starts in the middle of the grid.

3. Next, scale the fuel levels by a factor of $k^{-0.25}$.

Table 4 shows summary statistics for the initial fire configurations randomly generated in this experiment. This experiment was designed to explore the ability of the various approaches to consider how the allocation of suppression teams now will impact future reward received. Because the reward function may be very different outside the local region of the fire, the approaches may need to plan many steps into the future.

Table 4 **Grid 2 initial fire statistics**

	$k=9$	$k=17$	$k=25$
Mean cells burning	37.8	154.9	363.7
Maximum cells burning	69	224	487
Mean fuel level for burning cells	5.3	7.5	8.9
Fuel level for non-burnt cells	8	11	13

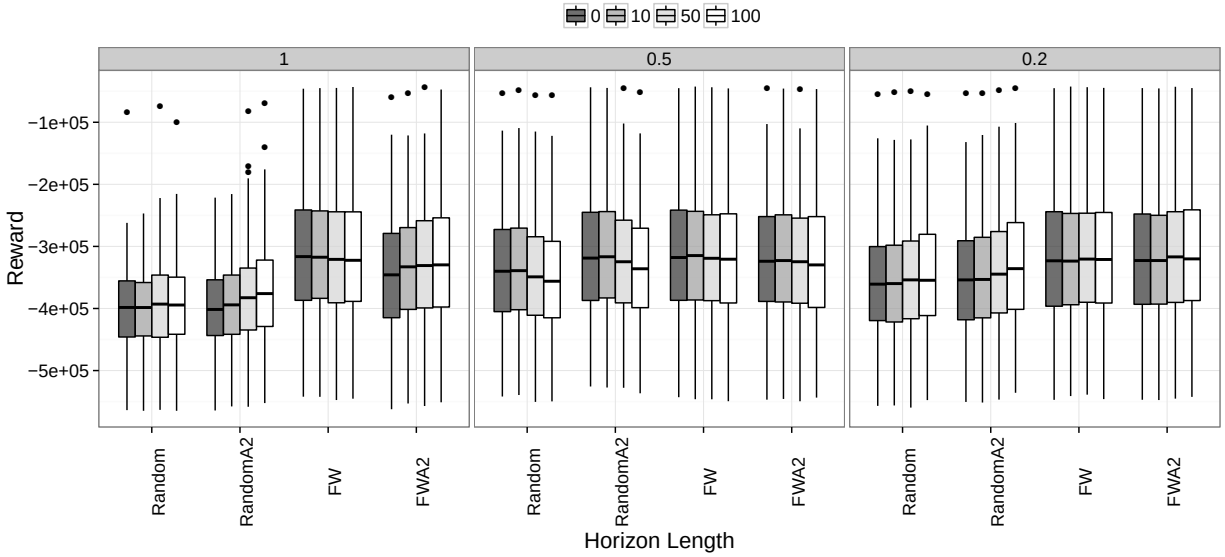
5.2. Tuning Hyperparameters for the MCTS Methodology

The MCTS approach includes a number of hyperparameters to control the performance of the algorithm. In this section, we explore the effects of some of these parameters using Grid 1, with $k = 20$ and 4 assets, and a time limit of 10 s per iteration. We vary the exploration bonus $c \in \{0, 10, 50, 100\}$, the progressive widening factors $\alpha = \alpha' \in \{1, 0.5, 0.2\}$, the depth $d \in \{1, 5, 10\}$, whether or not we use Algorithm 2 (based on genetic algorithm search heuristics) in action generation, and whether we use the random suppression heuristic or the FW heuristic in the rollout. For each combination of hyperparameters, we run 256 simulations and aggregate the results. Figure 6 presents box plots of the cumulative reward; for simplicity, we focus on depth $d = 5$, as the insights are qualitatively similar for the two other values of d. The top panel groups these box plots by the exploration bonus c, while the bottom one groups the same plots by the heuristic used. Several features are noticeable:

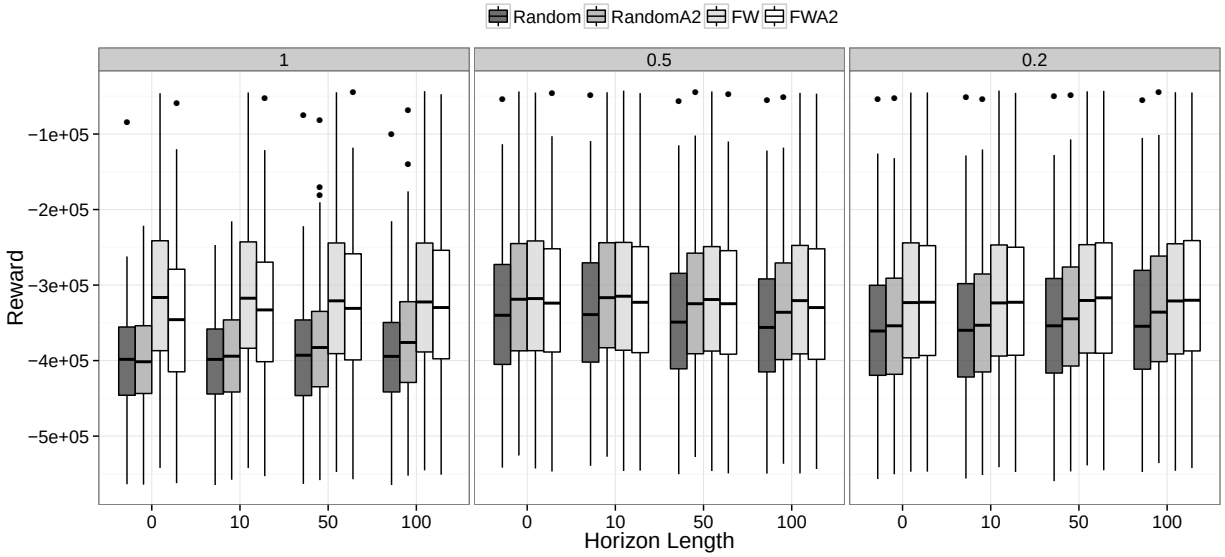
1. The effect of the exploration bonus c is small and depends on the heuristic used. For the FW heuristic, the effect is negligible. One explanation for this features is that the FW heuristic is fairly strong on it own. Hence, the benefits from local changes to this policy are somewhat limited. For the random suppression heuristic, there may be some effect, but it seems to vary with α. When $\alpha = 0.2$, increased exploration benefits the random suppression heuristic, but when $\alpha = 0.5$, less exploration is preferred.

2. From the second panel, in all cases it seems that MCTS with the FW heuristic outperforms MCTS with the random suppression heuristic. The benefit of Algorithm 2 in action generation, however, is less clear. Using Algorithm 2 does seem to improve the performance of the random suppression heuristic in some cases. For the FW heuristic, there seems to be little benefit. Again, one explanation of this phenomenon is that the FW heuristic already generates fairly good actions on its own. With only 10 s of computational time, it is difficult for the MCTS algorithm to find superior alternatives.

To assess the statistical significance of these differences, we fit a linear regression/additive effects model (Rice 2007). To simplify the analysis, we fit two separate models, one for the random suppression heuristic and one for the FW heuristic. Moreover, to simplify the models we use backward stepwise variable deletion beginning with the full model with interactions of order two (Hastie et al. 2009). Table 5 displays the results. In each column of values, the first value indicates the size of the effect, while the parenthesized value indicates the p-value of the effect (how statistically significant the effect is). For example, the effect for $\alpha = 0.2$ is 7292.33, which means that our estimated reward increases by 7292.33 when $\alpha = 0.2$ relative to the reward $\alpha = 0.1$. This effect is also quite statistically significant, as the p-value is very close to zero (0.03). In contrast, for $\alpha = 0.5$, the effect of -3073.68 has a much higher p-value (0.37) and so is less statistically significant. Note



(a)



(b)

Figure 6 **The cumulative reward of the MCTS algorithm for various values organized by the exploration parameter c and the rollout heuristic. "Random" and "FW" refer to the random burn and Floyd-Warshall heuristics. "A2" indicates that we additionally use our Algorithm 2 (based on genetic algorithm search heuristics) in the action generation phase.**

that there are also interaction effects; for example, when $\alpha = 0.5$ and $d = 5$, not only do we experience the -3073.68 effect from $\alpha = 0.5$ alone and the $+34551.16$ effect from $d = 5$ alone, but we also experience a joint effect of $+4412.72$ due to both α and d being 0.5 and 5 at the same time respectively.

Table 5 Estimated effects for the MCTS hyperparameters

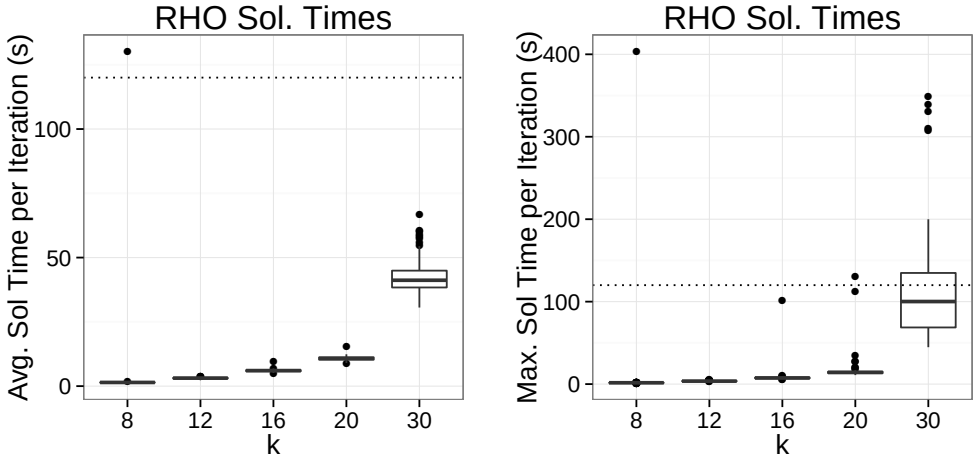
	FW Heuristic		Random Suppression	
(Intercept)	-353903.78^{***}	(0.00)	-408969.13^{***}	(0.00)
$\alpha = 0.5$	-3073.68	(0.37)	5335.56	(0.16)
$\alpha = 0.2$	7292.33^{*}	(0.03)	4616.83	(0.23)
Depth $= 5$	34551.16^{***}	(0.00)	4211.96	(0.14)
Depth $= 10$	35375.98^{***}	(0.00)	3952.83	(0.17)
A2	-40434.84^{***}	(0.00)	-976.72	(0.71)
$c = 10$			2857.04	(0.32)
$c = 50$			6900.73^{*}	(0.02)
$c = 100$			9366.90^{**}	(0.00)
$\alpha = 0.5$: Depth $= 5$	4412.72	(0.29)	58653.80^{***}	(0.00)
$\alpha = 0.2$: Depth $= 5$	-11279.75^{**}	(0.01)	41290.86^{***}	(0.00)
$\alpha = 0.5$: Depth $= 10$	2989.4	(0.48)	65456.71^{***}	(0.00)
$\alpha = 0.2$: Depth $= 10$	-11282.11^{**}	(0.01)	47508.01^{***}	(0.00)
$\alpha = 0.5$: A2	8467.03^{*}	(0.01)	6960.33^{*}	(0.02)
$\alpha = 0.2$: A2	20363.68^{***}	(0.00)	-1457.66	(0.61)
depth $= 5$: A2	23627.58^{***}	(0.00)		
depth $= 10$: A2	24100.29^{***}	(0.00)		
$\alpha = 0.5$: $c = 10$			-2543.39	(0.53)
$\alpha = 0.2$: $c = 10$			-983.12	(0.81)
$\alpha = 0.5$: $c = 50$			-10139.50^{*}	(0.01)
$\alpha = 0.2$: $c = 50$			-1250.75	(0.76)
$\alpha = 0.5$: $c = 100$			-16684.02^{***}	(0.00)
$\alpha = 0.2$: $c = 100$			-674.51	(0.87)
depth $= 5$: A2			12733.89^{***}	(0.00)
depth $= 10$: A2			12608.70^{***}	(0.00)
R^2	0.06		0.14	
adj. R^2	0.06		0.14	

† significant at $p < 0.10$; $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$
See Section 5.2 for details. Baseline should be interpreted as the value of $\alpha = 1$, Depth of 1, $c = 0$, without Algorithm 2 (based on genetic algorithm search heuristics).

One can check that the features we observed graphically from the box plots are indeed significant. Moreover, the random suppression heuristic demonstrates interesting second order effects between the depth and α. The performance improves substantially when the depth is greater than one and we restrict the size of the searched actions. One explanation is that both parameter serve to increase the quality of the search tree, i.e., its depth, and the accuracy of the estimate at each of the searched nodes.

5.3. State Space Size

We first study the performance of our algorithms as the size of the state space grows. We simulate the performance of each of our methods on Grid 1 with either 4 or 8 suppression teams, using our



(a) Average (b) Maximum

Figure 7 **Average and maximum iteration solution time with 8 teams and a desired time limit of 120 s (dotted line). Instances which exceed their allotted time are typically not solved to optimality.**

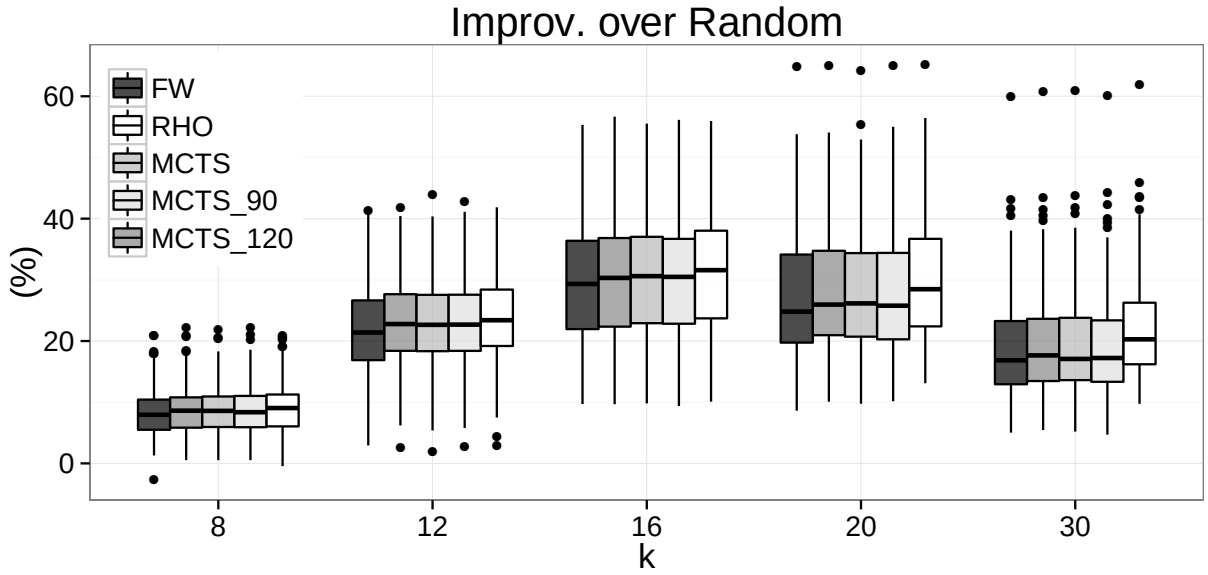
default values of the hyperparameters and varying $k \in \{8, 12, 16, 20, 30\}$. For each algorithm and combination of parameters, we simulate 256 runs and amalgamate the results.

Figures 7a and 7b show the average and maximum solution time per iteration of the RHO methodology when requesting at most 120 s of computation time. Notice that for most grids, the average time is well below the threshold – in these instances the underlying mixed integer program is solved to optimality. For some grids, though, there are a few iterations which require much longer to find a feasible integer solution (cf. the long upper-tail in Figure 7b). Consequently, we compare our RHO method to the MCTS method with 60 s, 90 s and 120 s of computation time.

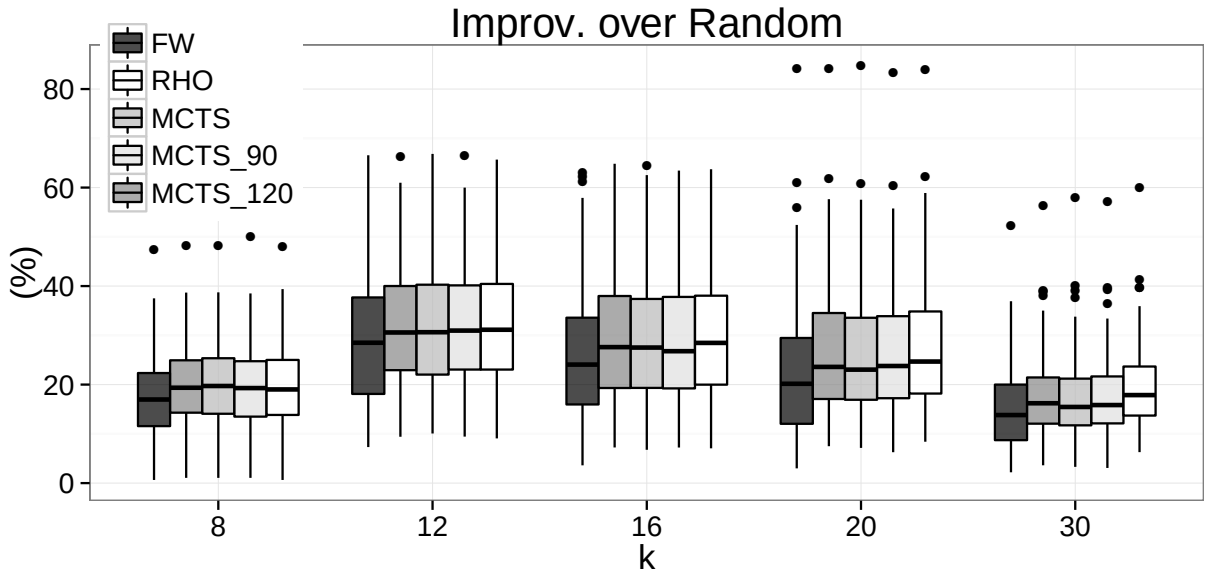
A summary of the results is seen in Figures 8a and 8b. Several features are evident in the plot. First, all three methods seem to outperform the FW heuristic, but there seems to only be a small difference between the three MCTS runs. The RHO method does seem to outperform MCTS method, especially for tail-hard instances. Namely, the lower whisker on the RHO plot is often shorter than the corresponding whisker on the MCTS plots.

To assess some of the statistical significance of these differences, we again fit two additive effects model; one for 8 suppression teams and one for 4. In both cases, there are no significant second order interactions. The coefficients of the first order interactions and corresponding p-values are shown in Table 6. The values suggest that the three MCTS methods are very similar and that the RHO method with a time limit of 60 s does outperform both.

In summary then, these results suggest that MCTS and RHO perform comparably for small state spaces, but as the size of the state space grows, RHO begins to outperform MCTS, with the magnitude of the edge growing with the state space size.



(a) 8 suppression teams.



(b) 4 suppression teams.

Figure 8 Performance as a function of state space size.

5.4. Action Space Size

In this set of experiments, we are interested in comparing the performance of the RHO and MCTS approaches as the size of the grid (and thus the state space) is fixed, and the number of resources (and thus the action space size) increases. We will see that as the action space size grows large, RHO performs increasingly better than MCTS; moreover, the rate of increase is greater for larger grids.

Table 6 Estimated effects for the percentage improvement relative to the random heuristic

	8 Teams		4 Teams	
	Coefficient	p-value	Coefficient	p-value
(Intercept)	7.80^{***}	(0.00)	16.94^{***}	(0.00)
$k=12$	14.48^{***}	(0.00)	12.02^{***}	(0.00)
$k=16$	21.89^{***}	(0.00)	9.76^{***}	(0.00)
$k=20$	19.48^{***}	(0.00)	6.08^{***}	(0.00)
$k=30$	10.86^{***}	(0.00)	-2.22^{***}	(0.00)
MCTS (120 s)	0.87^{*}	(0.03)	3.03^{***}	(0.00)
MCTS (90 s)	0.88^{*}	(0.03)	2.89^{***}	(0.00)
MCTS (60 s)	0.83^{*}	(0.04)	2.74^{***}	(0.00)
RHO	2.22^{***}	(0.00)	3.80^{***}	(0.00)
R^2	0.47		0.21	
adj. R^2	0.47		0.21	

† significant at $p<0.10$; $^{*}p<0.05$; $^{**}p<0.01$; $^{***}p<0.001$
For details, see Section 5.3. The intercept should be interpreted as baseline of $k=8$ with the FW heuristic.

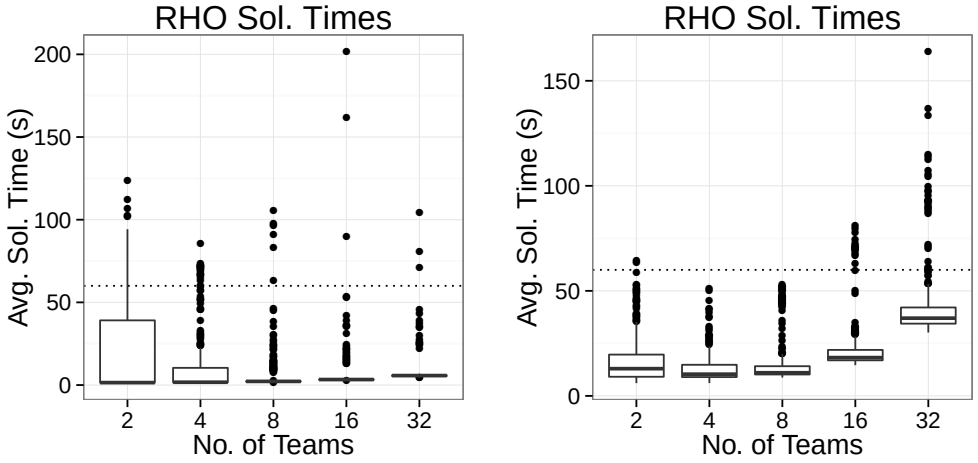
Intuition suggests that the performance of the MCTS algorithm is highly dependent on the magnitude of the action branching factor, i.e., the number of actions available from any given state. As discussed in Section 4, without progressive widening, when the action branching factor is larger than the number of iterations, the MCTS algorithm will only expand the search tree to depth one. Even with progressive widening, choosing good candidate actions is critical to growing the search tree in relevant directions. A simple calculation using Stirling's approximation confirms that for the MDP outlined in Section 2, the action branching factor at time t is given by

$$\frac{N_B(t)^{|\mathcal{I}|}}{|\mathcal{I}|!} \approx \frac{\left(e \cdot \frac{N_B(t)}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}{\sqrt{2\pi|\mathcal{I}|}}, \tag{13}$$

where $N_B(t)$ is the number of cells that are burning at time t. For even medium sized-instances, this number is extremely large. Consequently, in this section we study the performance of our algorithms with respect to the action branching factor.

We have already seen initial results in Section 5.2 suggesting that both our progressive widening and Algorithm 2 for action generation improve upon the base MCTS algorithm. It remains to see how MCTS with these refinements compares to our rolling horizon optimization approach. We compute the relative improvement of the MCTS and RHO approaches over the randomized suppression heuristic on Grid 1 over 256 simulations and $k=10$.

Recall that it is not possible to control for the exact time used by the RHO algorithm. Figure 9a shows a box-plot of the average time per iteration for the RHO approach. Based on this plot, we feel that comparing the results to the MCTS algorithm with 60 s of computational time is a fair comparison.



(a) $k = 10$ (b) $k = 20$

Figure 9 **RHO average solution times.**

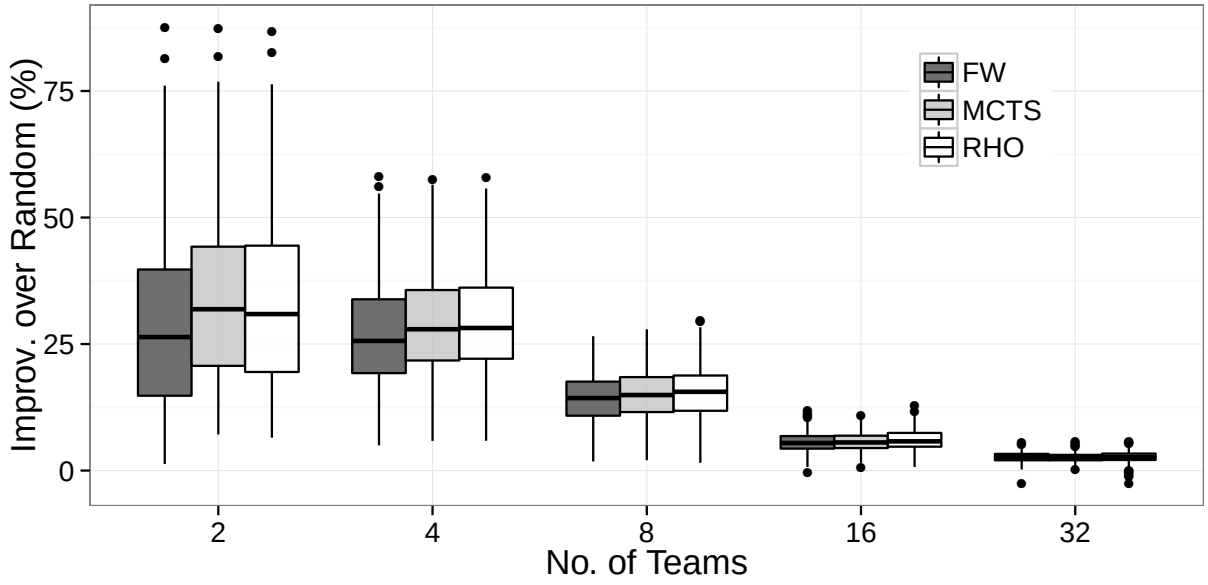


Figure 10 **Performance as function of number of suppression teams, $k = 10$.**

Figure 10 summarizes the average relative improvement for each our methods for $k = 10$ as the number of teams varies. From this plot, we can see that the relative performance of all our methods degrades as the number of teams become large. This is principally because the randomized suppression heuristic improves with more teams. Although the FW heuristic is clearly inferior, the remaining approaches appear to perform similarly. Indeed, ANOVA testing suggests the difference between RHO and MCTS is not statistically significant (p-values are well above 0.4).

To try and isolate more significant differences between the methodologies, we re-run the above experiment with $k = 20$. The results can be seen in Figure 11 and the average solution times in

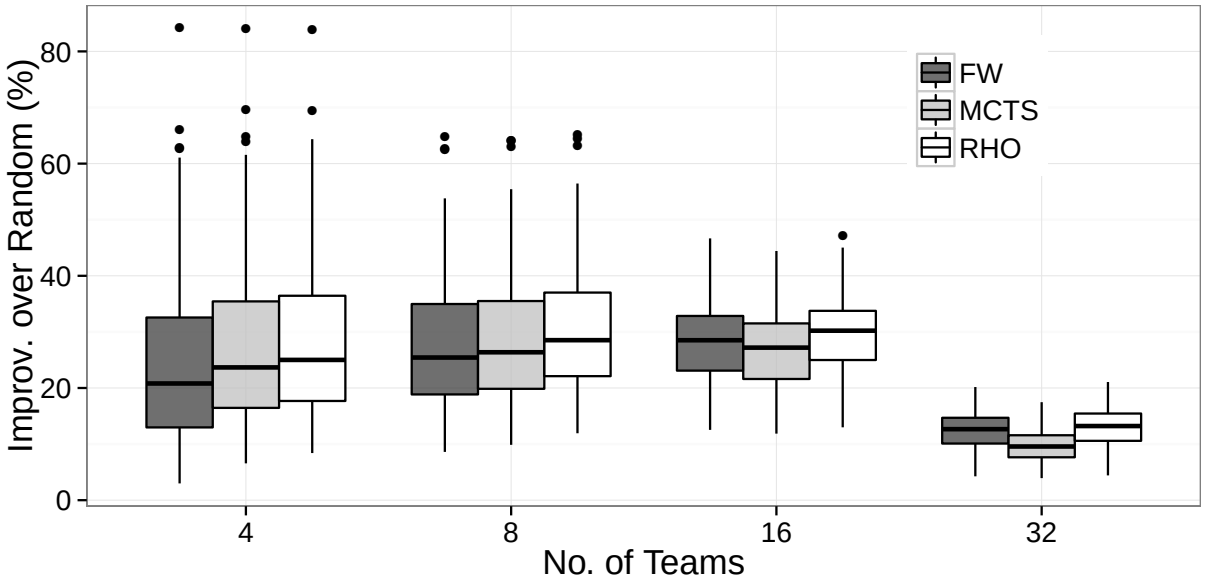


Figure 11 **Performance as a function of number of suppression teams, $k = 20$.**

Figure 9b. In contrast with the previous experiment, RHO appears to outperform MCTS. Interestingly, although MCTS seems to outperform the FW heuristic for a small number of suppression teams, it performs worse for more teams. To test the significance of these differences, we fit a linear regression model for the improvement over the randomized suppression heuristic as a function of the number of teams, the algorithm used, and potential interactions between the number of teams and the algorithm used. The results are in Table 7. The intercept value is the baseline, *fitted value of the FW heuristic policy for 4 teams.* Several observations can be drawn from Table 7. First, RHO outperforms FW for all choices of team size with statistical significance. MCTS, however, is statistically worse than FW with 16 or 32 teams.

In summary, differences between the RHO and MCTS methods become visible only when the grid size k is large, i.e., when the instances are sufficiently "difficult" to solve. It appears that although progressive widening and Algorithm 2 for action selection partially address the challenges of a large action state branching factor, the rolling horizon optimization approach is better suited to these instances.

5.5. Asymmetric Costs and Horizon Length

In this set of experiments, we are interested in understanding how sensitive RHO and MCTS are to their respective horizon length parameters and whether they are able to "plan for the long term" with regard to the cost structure. In particular, we examine the performance of our algorithms on Grid 2 (cf. Section 5.1). Recall that in this setup, the cost structure is asymmetric; cells to the left side of the grid are more important to suppress than cells to the right. At the same time, the

Table 7 Estimated effects for MCTS and RHO with varying team size

	Coefficients	p-value
(Intercept)	23.40^{***}	(0.00)
8 Teams	3.92^{***}	(0.00)
16 Teams	4.79^{***}	(0.00)
32 Teams	-10.84^{***}	(0.00)
MCTS	3.01^{***}	(0.00)
RHO	4.55^{***}	(0.00)
8 Teams : MCTS	$-1.93^{\dagger}$	(0.10)
16 Teams : MCTS	-4.36^{***}	(0.00)
32 Teams : MCTS	-5.86^{***}	(0.00)
8 Teams : RHO	-1.49	(0.20)
16 Teams : RHO	-3.19^{**}	(0.01)
32 Teams : RHO	-4.02^{***}	(0.00)
R^2	0.36	
adj. R^2	0.36	

† significant at $p < 0.10$; $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$
See also Section 5.4. The intercept should be interpreted as the performance of four teams under the FW algorithm.

local cost structure at the point of ignition is relatively flat. A good algorithm must recognize the differential cost despite the local cost structure. However, both RHO and MCTS are constrained in terms of how many steps into the future they account for (for RHO, this is controlled through the horizon length T and for MCTS, through the depth d).

We consider different combinations of $\lambda \in \{0.1, 0.2, 0.4\}$, $k \in \{9, 17, 25\}$, and horizon length in $\{2, 5, 10\}$. For each combination, we consider 256 simulations. Figure 12 summarizes the performances of each of our methods. For all the methods, there is an upward trend as λ increases. For small values of λ (i.e., flatter reward structure), RHO has a marked edge over the other methods. As λ increases, the difference shrinks. Figure 13 shows the same box plots, but grouped by method. This plot suggests that increasing the horizon length from 2 to 5 improves performance, but there is negligible improvement for length 10. This agrees to some extent with our intuition: as the horizon length increases, the algorithms become less myopic and are better able to account for the rapid growth in reward as one moves to the right of the grid.

We investigate the statistical significance of these differences by fitting additive effects models. The full model contains many insignificant interactions. We drop insignificant variables using backwards stepwise deletion. The resulting fit can be seen in Table 8. The fitted model suggests that differences observed in the previous plot are statistically significant.

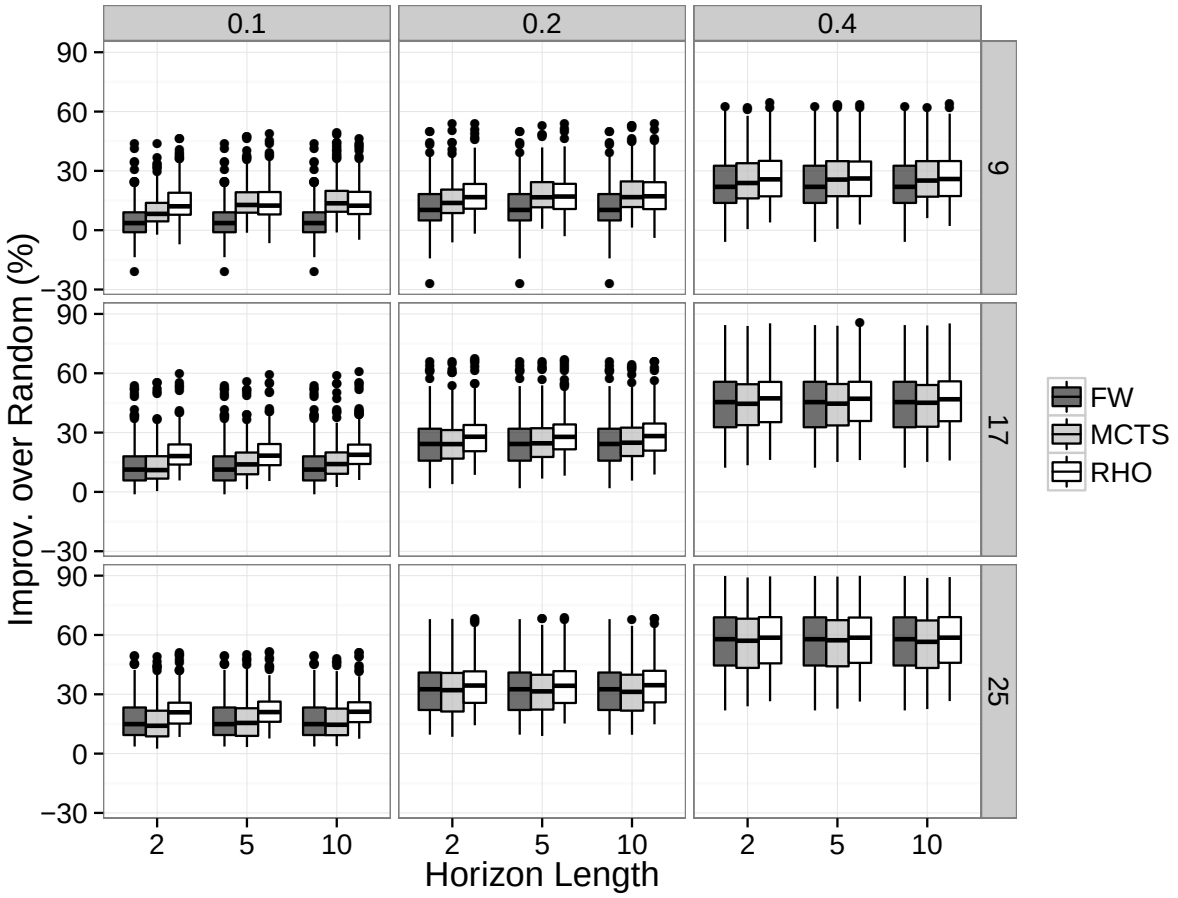


Figure 12 **Performance as a function of horizon length. Different panels correspond to different values of $k \in \{9, 17, 25\}$ and and $\lambda \in \{0.1, 0.2, 0.4\}$.**

6. Numerical Comparisons for Queueing Network Control

6.1. Background

For our experiments, we use the same custom implementation of MCTS in C++ used in Section 5 and for the fluid (rolling horizon optimization) approach, we use the results from Bertsimas et al. (2015), which were obtained using a custom implementation of the solution approach of Luo and Bertsimas (1998) in C. All experiments for MCTS were conducted on a 2.6GHz quad-core computer with 16GB of RAM in a single-threaded environment.

In Table 9, we specify the parameters of the MCTS approach. These parameters were chosen on the basis of preliminary testing that suggested good performance and comparable timing behavior to the fluid method. With regard to action generation, we consider a random action generation approach; for the rollout policy, we will consider the random policy and the $c\mu$ policy described in Section 4.2.2.

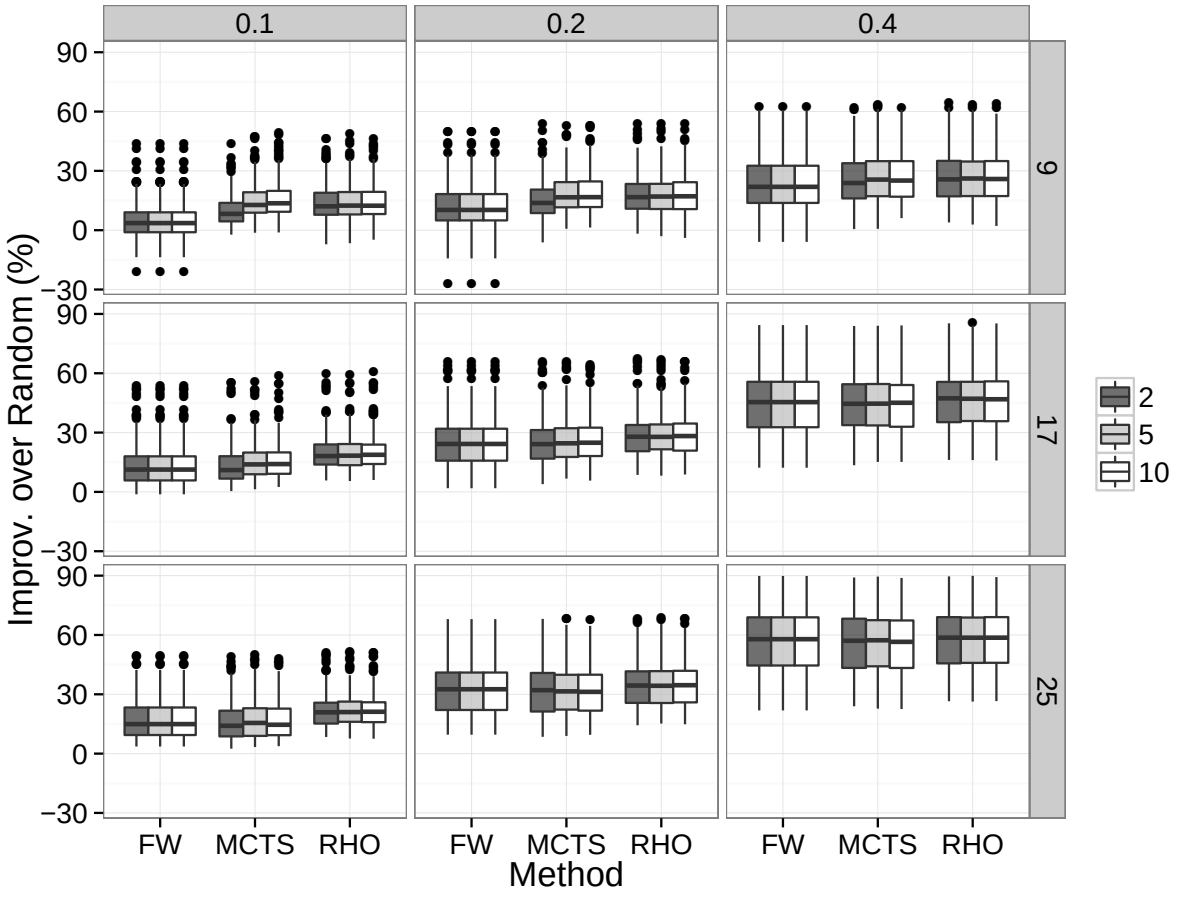


Figure 13 **Performance as a function of horizon length, grouped by method. Different panels correspond to different values of $k \in \{9, 17, 25\}$ and and $\lambda \in \{0.1, 0.2, 0.4\}$.**

In what follows, we will compare the MCTS and the fluid optimization approaches on the four example network studied in Bertsimas et al. (2015). The insights that emerge from this computational study can be summarized as follows:

- MCTS and the fluid approach perform comparably on the criss-cross network (two servers, three job classes) and six-class network (two servers, three job classes per server).
- For the extended six-class network, MCTS tends to slightly outperform the fluid approach, while for the reentrant network, the fluid approach significantly outperforms MCTS. Overall, it seems that on large networks with complicated topologies, the fluid approach exhibits an edge over MCTS.
- In general, MCTS with the random rollout is able to significantly improve on the random policy, but MCTS with the $c\mu$ rollout is unable to improve on the basic $c\mu$ policy.

Table 8 Estimated effects for Section 5.5

Effect	Coefficient	p-value
(Intercept)	5.45^{***}	(0.00)
$\lambda = 0.2$	6.66^{***}	(0.00)
$\lambda = 0.4$	17.70^{***}	(0.00)
$k = 17$	7.84^{***}	(0.00)
$k = 25$	10.87^{***}	(0.00)
Depth 5	0.00	(1.00)
Depth 10	0.00	(1.00)
MCTS	6.25^{***}	(0.00)
RHO	8.76^{***}	(0.00)
$\lambda = 0.2 : k = 17$	5.22^{***}	(0.00)
$\lambda = 0.4 : k = 17$	14.38^{***}	(0.00)
$\lambda = 0.2 : k = 25$	9.33^{***}	(0.00)
$\lambda = 0.4 : k = 25$	23.56^{***}	(0.00)
$\lambda = 0.2$: MCTS	-1.47^{**}	(0.00)
$\lambda = 0.4$: MCTS	-2.94^{***}	(0.00)
$\lambda = 0.2$: RHO	-2.81^{***}	(0.00)
$\lambda = 0.4$: RHO	-4.98^{***}	(0.00)
$k = 17$: MCTS	-5.14^{***}	(0.00)
$k = 25$: MCTS	-6.51^{***}	(0.00)
$k = 17$: RHO	-2.33^{***}	(0.00)
$k = 25$: RHO	-3.48^{***}	(0.00)
Depth 5 : MCTS	1.50^{**}	(0.00)
Depth 10 : MCTS	1.47^{**}	(0.00)
Depth 5 : RHO	0.18	(0.72)
Depth 10 : RHO	0.14	(0.78)

† significant at $p < 0.10$; $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$
The intercept should be interpreted as the value of the FW heuristic when $\lambda = .1$, $k = 9$ and the horizon length is 2.

Table 9 Default parameters for MCTS

Parameter	Value
Num. trajectories	20
Exploration bonus	20
Depth	10
Progressive widening, action space α	0.01
Progressive widening, state space α'	0.01
Progressive widening, action space k	5
Progressive widening, state space k'	5

6.2. Criss-cross network

The first network that we study is the example criss-cross network shown in Figure 2 in Section 2.2. We consider six different sets of values for the arrival rates λ_1 and λ_2 and the service rates μ_1, μ_2 and μ_3, shown in Table 10; for further details on these parameter sets the reader is referred to Bertsimas et al. (2015).

Table 10 Parameter sets for the criss-cross network.

Parameter set	λ_1	λ_2	μ_1	μ_2	μ_3
I.L.	0.3	0.3	2	2	1.5
B.L.	0.3	0.3	2	2	1
I.M.	0.6	0.6	2	2	1.5
B.M.	0.6	0.6	2	2	1
I.H.	0.9	0.9	2	2	1.5
B.H.	0.9	0.9	2	2	1

Table 11 Average long-run number of jobs in system for the criss-cross queueing network under different methods.

Parameter set	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
I.L.	0.678	0.679	0.681	0.677	0.678
B.L.	0.858	0.859	0.856	0.856	0.857
I.M.	2.167	2.170	2.168	2.160	2.162
B.M.	3.005	3.006	3.012	2.983	2.965
I.H.	10.359	10.470	10.430	10.546	10.398
B.H.	18.157	18.097	17.910	18.229	18.430

Table 13 displays the results. For this network, we can see that the methods are in general comparable and achieve very similar performance. In the **B.H.** case, both MCTS methods outperform the fluid method – for example, MCTS with random rollout attains a value of 17.910 compared to 18.430 for the fluid method, which is a difference of over 2% relative to the fluid method. In general, each MCTS variant does not appear to perform significantly differently from its underlying rollout policy; in some cases the MCTS method performs worse than its underlying rollout policy (e.g., MCTS-$c\mu$ for **I.H.** performs worse than $c\mu$ for the same parameter set), while in some cases MCTS offers a considerable improvement over the underlying rollout (e.g., MCTS-random for **B.H.** performs better than the random policy alone).

6.3. Six-class network

In this experiment, we consider the six-class network from Bertsimas et al. (2015). The structure of the network is shown in Figure 14. We consider six different sets of parameter values given in Table 12; as with the criss-cross network, more details on these parameters can be found in Bertsimas et al. (2015).

Table 13 displays the results. From this table, we can see that the fluid method generally performs better than MCTS. The most significant example is **B.H.** where the fluid method achieves an average number of jobs of 15.670 compared to 18.425 and 17.882 for MCTS-random and MCTS-$c\mu$ respectively. In general, MCTS with the random rollout significantly outperforms the random policy, but MCTS with the $c\mu$ rollout does not always improve on the performance of the $c\mu$ policy on its own and in general is slightly worse.

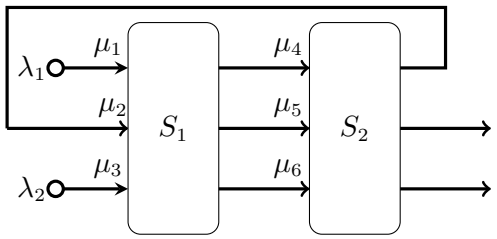


Figure 14 **Six-class network.**

Table 12 **Parameters for six-class queueing network.**

Parameter set	λ_1	λ_2	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6
I.L.	3/140	3/140	1/8	1/2	1/4	1/4	3/14	2/3
B.L.	3/140	3/140	1/8	1/2	1/4	1/6	1/7	1
I.M.	6/140	6/140	1/8	1/2	1/4	1/4	3/14	2/3
B.M.	6/140	6/140	1/8	1/2	1/4	1/6	1/7	1
I.H.	9/140	9/140	1/8	1/2	1/4	1/4	3/14	2/3
B.H.	9/140	9/140	1/8	1/2	1/4	1/6	1/7	1

Table 13 **Average long-run number of jobs in system for six-class queueing network under different methods.**

Parameter set	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
I.L.	0.780	0.748	0.756	0.757	0.750
B.L.	0.965	0.928	0.938	0.939	0.923
I.M.	2.728	2.355	2.433	2.442	2.301
B.M.	3.807	3.235	3.319	3.320	3.024
I.H.	14.654	9.452	9.561	9.736	9.435
B.H.	24.695	18.167	18.425	17.882	15.670

6.4. Extended six-class network

We now study a more complicated extension of the six class network. In particular, we assume that the servers repeat, as shown in Figure 15. We assume that each server is associated with 3 classes (i.e., in Figure 15, $n = 3$). The only external arrivals to the system are at classes 1 and 3, and the arrival rates for those classes are the same as in parameter set **B.H.** in Table 12 for the six-class network. The service rates for the classes also follow parameter set **B.H.** in Table 12 in the following way: for odd values of j, $\mu_{n(j-1)+1} = \mu_1$, $\mu_{n(j-1)+2} = \mu_2$ and $\mu_{n(j-1)+3} = \mu_3$, while for even values of j, $\mu_{n(j-1)+1} = \mu_4$, $\mu_{n(j-1)+2} = \mu_5$ and $\mu_{n(j-1)+3} = \mu_6$.

Table 14 compares the different methods as the number of servers m varies from 2 to 7. From this table, we can see that aside from $m = 2$ and $m = 7$, MCTS with either the random or the $c\mu$ rollout policy performs better than the fluid method; for $m = 2$ and $m = 7$, the fluid method performs better. As in the six-class network, MCTS-random significantly improves on the random policy, but MCTS-$c\mu$ generally performs slightly worse than the basic $c\mu$ policy. It turns out for

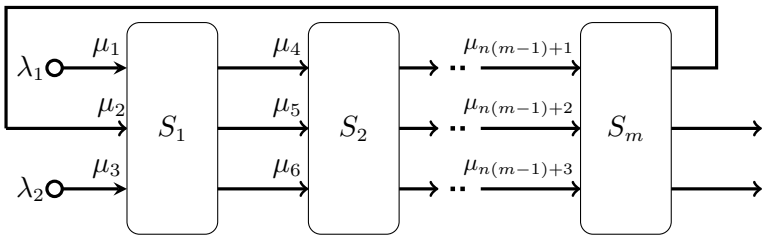


Figure 15 **The extended six-class network.**

Table 14 **Average long-run number of jobs in system for extended six-class queueing network under different methods.**

Num. servers m	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
2	24.315	18.296	19.005	17.948	15.422
3	38.891	25.425	27.784	25.861	26.140
4	51.453	35.721	36.060	36.452	38.085
5	67.118	43.314	44.905	45.727	45.962
6	78.830	53.801	55.581	55.315	56.857
7	91.218	60.743	65.638	66.614	64.713

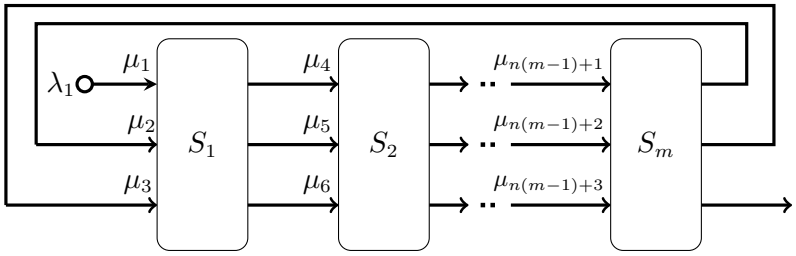


Figure 16 **Reentrant queueing network.**

this network that the $c\mu$ policy, with the exception of $m = 2$, generally performs the best of all the methods.

6.5. Reentrant network

Finally, we consider a reentrant network, where after a job passes through the m servers, it re-enters the system at the first server and is processed again two more times (as a class 2 job and a class 3 job) before exiting the system. Figure 16 displays the topology of this queueing network. The arrival rate of class 1 and the service rates of the classes are the same as for the extended six-class network of the previous section. We vary the number of servers from $m = 2$ to $m = 7$.

Table 15 displays the average number of jobs in the system under the different types of policies as the number of servers m varies from 2 to 7. For all values of m, the fluid policy performs the best, and significantly outperforms both versions of MCTS. For example, for $m = 7$, MCTS-$c\mu$ achieves a value of 63.696 while the fluid method achieves a value of 54.711 – a difference of over 14% relative to MCTS-$c\mu$. As in the extended six-class network, MCTS-random is able to provide a

Table 15 **Average long-run number of jobs in system for reentrant queueing network under different methods.**

Num. servers m	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
2	25.076	18.199	17.991	17.796	15.422
3	37.908	26.937	27.233	26.257	25.955
4	50.940	34.934	37.194	36.734	32.014
5	64.273	42.387	45.629	44.866	40.113
6	79.165	51.945	57.189	54.682	48.781
7	92.672	59.958	67.531	63.696	54.711

significant improvement over the baseline random policy, but MCTS-$c\mu$ generally does not improve on the basic $c\mu$ policy, especially for higher values of m.

7. Conclusion

In this study, we consider two dynamic resource allocation problems: one concerning tactical wildfire management, in which fire spreads stochastically on a finite grid and the decision maker must allocate resources at discrete epochs to suppress it, and one involving queueing network control, where a decision maker must determine which jobs are to be served by the servers as jobs arrive and pass through the network. We study two different solution approaches: one based on Monte Carlo tree search, and one based on rolling horizon optimization.

Our study makes two broad methodological contributions. The first of these contributions is to the understanding of MCTS: to the best of our knowledge, this study is the first application of Monte Carlo tree search to high-dimensional dynamic resource allocation problems. Our numerical results uncover some interesting insights into how MCTS behaves in relation to parameters such as the exploration bonus, the progressive widening parameters and others, as well as larger components such as the method of action generation and the rollout heuristic. Our results show that these components are highly interdependent and cannot be calibrated separately of each other—for example, our results show that the choices of action generation method and progressive widening factor become very important when the rollout heuristic is not strong on its own (e.g., a random heuristic) but are less valuable when the rollout heuristic is strong to begin with (e.g., the Floyd-Warshall heuristic in the case of the wildfire management problem). These insights will be valuable for practitioners interested in applying MCTS to other problems.

The second broad methodological contribution is towards the understanding of the relative merits of MCTS and RHO. Our results show that while both methodologies exhibit comparable performance for smaller instances, for larger instances, the mathematical optimization approach exhibits a significant edge. Initial evidence suggests this edge may be related more closely to action branching factor than the state space branching factor.

Acknowledgments

This work is sponsored by the Assistant Secretary of Defense for Research and Engineering, ASD(R&E), under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government. The work of the fifth author was supported by a PGS-D award from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

Acimovic, J, S Graves. 2012. Making better fulfillment decisions on the fly in an online retail environment. Tech. rep., Working Paper, Massachusetts Institute of Technology, Boston, MA.

Arneson, Broderick, Ryan Hayward, Philip Henderson. 2009. MoHex wins Hex tournament. *International Computer Games Association Journal* **32** 114–116.

Arneson, Broderick, Ryan B Hayward, Philip Henderson. 2010. Monte Carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games* **2** 251–258.

Auer, Peter, Nicolò Cesa-Bianchi, Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* **47** 235–256.

Avram, Florin, Dimitris Bertsimas, Michael Ricard. 1995. Fluid models of sequencing problems in open queueing networks: An optimal control approach. *Institute for Mathematics and its Applications* 199–234.

Barnhart, Cynthia, Peter Belobaba, Amedeo R Odoni. 2003. Applications of operations research in the air transport industry. *Transportation science* **37** 368–391.

Bellman, Richard Erneset. 1957. *Dynamic Programming*. Princeton University Press.

Bemporad, Alberto. 2006. Model predictive control design: New trends and tools. *Decision and Control, 2006 45th IEEE Conference on*. IEEE, 6678–6683.

Ben-Tal, Aharon, Laurent El Ghaoui, Arkadi Nemirovski. 2009. *Robust optimization*. Princeton University Press.

Bertsekas, Dimitri P. 1995. *Dynamic programming and optimal control*, vol. 1. Athena Scientific Belmont.

Bertsekas, Dimitri P, Robert G Gallager. 1992. *Data networks*. Prentice-hall Englewood Cliffs, NJ.

Bertsimas, Dimitris, David B Brown, Constantine Caramanis. 2011. Theory and applications of robust optimization. *SIAM review* **53** 464–501.

Bertsimas, Dimitris, Shubham Gupta, Guglielmo Lulli. 2014. Dynamic resource allocation: A flexible and tractable modeling framework. *European Journal of Operational Research* **236** 14–26.

Bertsimas, Dimitris, Ebrahim Nasrabadi, Ioannis Ch. Paschalidis. 2015. Robust fluid processing networks. *IEEE Transactions on Automatic Control* **60** 715–728.

Bertsimas, Dimitris, Ioannis Ch Paschalidis, John N Tsitsiklis. 1994. Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *The Annals of Applied Probability* 43–75.

Bertsimas, Dimitris, Ioannis Ch Paschalidis, John N Tsitsiklis. 1995. Branching bandits and klimov's problem: Achievable region and side constraints. *IEEE Transactions on Automatic Control* **40** 2063–2075.

Bertsimas, Dimitris, Sarah Stock Patterson. 1998. The air traffic flow management problem with enroute capacities. *Operations Research* **46** 406–422.

Bhat, Nikhil, Vivek Farias, Ciamac C Moallemi. 2012. Non-parametric approximate dynamic programming via the kernel method. *Advances in Neural Information Processing Systems (NIPS)*. 386–394.

Boychuck, D., W. J. Braun, R. J. Kulperger, Z. L. Krougly, D. A. Stanford. 2008. A stochastic forest fire growth model. *Environmental and Ecological Statistics* **1** 1–19.

Bracmort, Kelsi. 2013. Wildfire management: Federal funding and related statistics. Tech. rep., Congressional Research Service. URL `https://www.fas.org/sgp/crs/misc/R43077.pdf`.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* **4** 1–43.

Buyukkoc, C, P Varaiya, J Walrand. 1985. The cμ rule revisited. *Advances in Applied pProbability* **17** 237–238.

Buzacott, John A, J George Shanthikumar. 1993. *Stochastic models of manufacturing systems*. Prentice Hall Englewood Cliffs, NJ.

Charnes, Abraham, William W Cooper. 1959. Chance-constrained programming. *Management science* **6** 73–79.

Chen, Hong, David D Yao. 1993. Dynamic scheduling of a multiclass fluid network. *Operations Research* **41** 1104–1115.

Ciancarini, Paolo, Gian Piero Favini. 2010. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence* **174** 670 – 684. doi:http://dx.doi.org/10.1016/j.artint.2010.04.017. URL `http://www.sciencedirect.com/science/article/pii/S0004370210000536`.

Ciocan, Dragos Florin, Vivek Farias. 2012. Model predictive control for dynamic resource allocation. *Mathematics of Operations Research* **37** 501–525.

Coffman Jr, EG, I Mitrani. 1980. A characterization of waiting time performance realizable by single-server queues. *Operations Research* **28** 810–821.

Coquelin, Pierre-Arnaud, Rémi Munos. 2007a. Bandit algorithms for tree search. *arXiv preprint cs/0703062* .

Coquelin, Pierre-Arnuad, Remi Munos. 2007b. Bandit algorithms for tree search. *Proceedings of the Twenty-Third Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-07)*. AUAI Press, Corvallis, Oregon, 67–74.

Couëtoux, Adrien, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, Nicolas Bonnard. 2011. Continuous upper confidence trees. *International Conference on Learning and Intelligent Optimization*. 433–445. doi:10.1007/978-3-642-25566-3_32.

Coulom, Rémi. 2007. Efficient selectivity and backup operators in monte-carlo tree search. *Computers and Games*. Springer, 72–83.

Dai, Jim G. 1995. On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *The Annals of Applied Probability* 49–77.

de Farias, Daniela Pucci, Benjamin Van Roy. 2003. The linear programming approach to approximate dynamic programming. *Operations Research* **51** 850–865.

Domshlak, Carmel, Zohar Feldman. 2013. To UCT, or not to UCT? *Sixth Annual Symposium on Combinatorial Search*.

Enzenberger, Markus, Martin Muller, Broderick Arneson, Richard Segal. 2010. Fuego—an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* **2** 259–270.

Erdelyi, Alexander, Huseyin Topaloglu. 2010. Approximate dynamic programming for dynamic capacity allocation with multiple priority levels. *IIE Transactions* **43** 129–142. doi:10.1080/0740817X.2010.504690. URL `http://www.tandfonline.com/doi/abs/10.1080/0740817X.2010.504690`.

Eyerich, Patrick, Thomas Keller, Malte Helmert. 2010. High-quality policies for the canadian traveler's problem. *Third Annual Symposium on Combinatorial Search*.

Finney, Mark Arnold. 2004. FARSITE: Fire area simulator–model development and evaluation. Research Paper RMRS-RP-4, USDA Forest Service.

Fleischer, Lisa, Jay Sethuraman. 2005. Efficient algorithms for separated continuous linear programs: the multicommodity flow problem with holding costs and extensions. *Mathematics of Operations Research* **30** 916–938.

Floyd, Robert W. 1962. Algorithm 97: Shortest path. *Communications of the ACM* **5** 345. doi:10.1145/367766.368168.

Fried, J. S., J. K. Gilless, J. Spero. 2006. Analysing initial attack on wildland fires using stochastic simulation. *International Journal of Wildland Fire* **15** 137–146. doi:http://dx.doi.org/10.1071/WF05027.

Gallego, Guillermo, Garrett van Ryzin. 1994. Optimal dynamic pricing of inventories with stochastic demand over finite horizons. *Management Science* **40** 999–1020.

Gelly, Sylvain, David Silver. 2011. Monte-Carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence* **175** 1856 – 1875. doi:http://dx.doi.org/10.1016/j.artint.2011.03.007.

Gorte, Ross. 2013. The rising cost of wildfire protection. Tech. rep., Headwater Economics. URL `http://headwaterseconomics.org/wphw/wp-content/uploads/fire-costs-background-report.pdf`.

Grieder, Pascal, Pablo A Parrilo, Manfred Morari. 2003. Robust receding horizon control-analysis & synthesis. *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1. IEEE, 941–946.

Gurobi Optimization, Inc. 2013. Gurobi optimizer reference manual. URL `http://www.gurobi.com`.

Harchol-Balter, Mor. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press.

Harrison, J Michael. 1988. Brownian models of queueing networks with heterogeneous customer populations. *Stochastic differential systems, stochastic control theory and applications*. Springer, 147–186.

Harrison, J Michael, Lawrence M Wein. 1990. Scheduling networks of queues: heavy traffic analysis of a two-station closed network. *Operations Research* **38** 1052–1064.

Hastie, Trevor, Robert Tibshirani, Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer.

Hu, Xiaolin, Lewis Ntaimo. 2009. Integrated simulation and optimization for wildfire containment. *ACM Trans. Model. Comput. Simul.* **19** 19:1–19:29. doi:10.1145/1596519.1596524. URL `http://doi.acm.org/10.1145/1596519.1596524`.

Hu, Xiaolin, Yi Sun, Lewis Ntaimo. 2012. Devs-fire: design and application of formal discrete event wildfire spread and suppression models. *SIMULATION* **88** 259–279. doi:10.1177/0037549711414592. URL `http://sim.sagepub.com/content/88/3/259.abstract`.

Jasin, Stefanus, Sunil Kumar. 2012. A re-solving heuristic with bounded revenue loss for network revenue management with customer choice. *Mathematics of Operations Research* **37** 313–345.

Kocsis, Levente, Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. *Proceedings of the European Conference on Machine Learning*. Springer, 282–293.

Lee, Chang-Shing, Mei-Hui Wang, Guillaume Chaslot, J-B Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, Tzung-Pei Hong. 2009. The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games* **1** 73–89.

Luo, Xiaodong, Dimitris Bertsimas. 1998. A new algorithm for state-constrained separated continuous linear programs. *SIAM Journal on control and optimization* **37** 177–210.

Maglaras, Constantinos, Joern Meissner. 2006. Dynamic pricing strategies for multiproduct revenue management problems. *Manufacturing & Service Operations Management* **8** 136–148.

Martin-Fernndez, Susana, Eugenio Martnez-falero, J. Manuel Prez-Gonzlez. 2002. Optimization of the resources management in fighting wildfires. *Environmental Management* **30** 352–364. doi:10.1007/s00267-002-2430-3.

Meyn, Sean P. 2003. Sequencing and routing in multiclass queueing networks part ii: Workload relaxations. *SIAM Journal on Control and Optimization* **42** 178–217.

Mišić, Bratislav, Olaf Sporns, Anthony R McIntosh. 2014. Communication efficiency and congestion of signal traffic in large-scale brain networks. *PLoS Computational Biology* **10** e1003427:1–10.

Nilim, Arnab, Laurent El Ghaoui. 2005. Robust control of markov decision processes with uncertain transition matrices. *Operations Research* **53** 780–798.

Ntaimo, Lewis, Julin A. Gallego Arrubla, Curt Stripling, Joshua Young, Thomas Spencer. 2012. A stochastic programming standard response model for wildfire initial attack planning. *Canadian Journal of Forest Research* **42** 987–1001. doi:10.1139/x2012-032.

Ntaimo, Lewis, Julian A. Gallego-Arrubla, Gan Jianbang, Curt Stripling, Joshua Young, Thomas Spencer. 2013. A simulation and stochastic integer programming approach to wildfire initial attack planning. *Forest Science* **59** 105 – 117. URL `http://search.ebscohost.com/login.aspx?direct=true&db=egh&AN=85825800&site=ehost-live`.

Papadimitriou, Christos H, John N Tsitsiklis. 1999. The complexity of optimal queuing network control. *Mathematics of Operations Research* **24** 293–305.

Paschalidis, Ioannis Ch, Chang Su, Michael C Caramanis. 2004. Target-pursuing scheduling and routing policies for multiclass queueing networks. *IEEE Transactions on Automatic Control* **49** 1709–1722.

Powell, Warren B, Joel A Shapiro, Hugo P Simão. 2002. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. *Transportation Science* **36** 231–249.

Powell, William B. 2011. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. 2nd ed. Wiley.

Pullan, Malcolm C. 1993. An algorithm for a class of continuous linear programs. *SIAM Journal on Control and Optimization* **31** 1558–1577.

Pullan, Malcolm C. 1995. Forms of optimal solutions for separated continuous linear programs. *SIAM Journal on Control and Optimization* **33** 1952–1977.

Pullan, Malcolm C. 1996. A duality theory for separated continuous linear programs. *SIAM Journal on Control and Optimization* **34** 931–965.

Reiman, Martin I, Qiong Wang. 2008. An asymptotically optimal policy for a quantity-based network revenue management problem. *Mathematics of Operations Research* **33** 257–282.

Rice, John A. 2007. *Mathematical Statistics and Data Analysis*. Cengage Learning.

Rothermel, Richard C. 1972. A mathematical model for predicting fire spread in wildland fuels. Tech. Rep. INT-115, USDA Forest Service.

Rubin, Jonathan, Ian Watson. 2011. Computer poker: A review. *Artificial Intelligence* **175** 958 – 987. doi:http://dx.doi.org/10.1016/j.artint.2010.12.005.

Shah, Devavrat, Damon Wischik. 2012. Switched networks with maximum weight policies: Fluid approximation and multiplicative state space collapse. *The Annals of Applied Probability* **22** 70–127.

Talluri, Kalyan, Garrett van Ryzin. 1998. An analysis of bid-price controls for network revenue management. *Management Science* **44** 1577–1593.

Talluri, Kalyan T, Garrett J van Ryzin. 2004. *The Theory and Practice of Revenue Management. International Series in Operations Research and Management Science, vol. 68*. Springer.

Tymstra, C., R.W. Bryce, B.M. Wotton, S.W. Taylor, O.B. Armitage. 2010. Development and structure of prometheus: The Canadian wildland fire growth simulation model. Information Report NOR-X-417, Canadian Forest Service.

Van Mieghem, Jan A. 1995. Dynamic scheduling with convex delay costs: The generalized c— mu rule. *Annals of Applied Probability* **5** 809–833.

Western Forestry Leadership Coalition. 2010. The true cost of wildfire in the Western U.S. URL `http://www.wflccenter.org/news_pdf/324_pdf.pdf`.

Whitley, Darrell. 1994. A genetic algorithm tutorial. *Statistics and Computing* **4** 65–85. doi:10.1007/BF00175354.